W3School PHP 参考手册

wizardforcel

Published with GitBook



目錄

W3School PHP 参考手册

来源：PHP 参考手册

整理：飞龙

PHP Array 函数

PHP Array 简介

array 函数允许您对数组进行操作。

PHP 支持单维和多维的数组。同时提供了用数据库查询结果来构造数组的函数。

安装

array 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP Array 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
array()	创建数组。	3
array_change_key_case()	返回其键均为大写或小写的数组。	4
array_chunk()	把一个数组分割为新的数组块。	4
array_combine()	通过合并两个数组来创建一个新数组。	5
array_count_values()	用于统计数组中所有值出现的次数。	4
array_diff()	返回两个数组的差集数组。	4
array_diff_assoc()	比较键名和键值，并返回两个数组的差集数组。	4
array_diff_key()	比较键名，并返回两个数组的差集数组。	5
array_diff_uassoc()	通过用户提供的回调函数做索引检查来计算数组的差集。	5
array_diff_ukey()	用回调函数对键名比较计算数组的差集。	5
array_fill()	用给定的值填充数组。	4
array_filter()	用回调函数过滤数组中的元素。	4
array_flip()	交换数组中的键和值。	4
array_intersect()	计算数组的交集。	4
array_intersect_assoc()	比较键名和键值，并返回两个数组的交集数组。	4
array_intersect_key()	使用键名比较计算数组的交集。	5

array_intersect_uassoc()	带索引检查计算数组的交集，用回调函数比较索引。	5
array_intersect_ukey()	用回调函数比较键名来计算数组的交集。	5
array_key_exists()	检查给定的键名或索引是否存在于数组中。	4
array_keys()	返回数组中所有的键名。	4
array_map()	将回调函数作用到给定数组的单元上。	4
array_merge()	把一个或多个数组合并为一个数组。	4
array_merge_recursive()	递归地合并一个或多个数组。	4
array_multisort()	对多个数组或多维数组进行排序。	4
array_pad()	用值将数组填补到指定长度。	4
array_pop()	将数组最后一个单元弹出（出栈）。	4
array_product()	计算数组中所有值的乘积。	5
array_push()	将一个或多个单元（元素）压入数组的末尾（入栈）。	4
array_rand()	从数组中随机选出一个或多个元素，并返回。	4
array_reduce()	用回调函数迭代地将数组简化为单一的值。	4
array_reverse()	将原数组中的元素顺序翻转，创建新的数组并返回。	4
array_search()	在数组中搜索给定的值，如果成功则返回相应的键名。	4
array_shift()	删除数组中的第一个元素，并返回被删除元素的值。	4
array_slice()	在数组中根据条件取出一段值，并返回。	4
array_splice()	把数组中的一部分去掉并用其它值取代。	4
array_sum()	计算数组中所有值的和。	4
array_udiff()	用回调函数比较数据来计算数组的差集。	5
array_udiff_assoc()	带索引检查计算数组的差集，用回调函数比较数据。	5
array_udiff_uassoc()	带索引检查计算数组的差集，用回调函数比较数据和索引。	5
array_uintersect()	计算数组的交集，用回调函数比较数据。	5
array_uintersect_assoc()	带索引检查计算数组的交集，用回调函数比较数据。	5
array_uintersect_uassoc()	带索引检查计算数组的交集，用回调函数比较数据和索引。	5

array_unique()	删除数组中重复的值。	4
array_unshift()	在数组开头插入一个或多个元素。	4
array_values()	返回数组中所有的值。	4
array_walk()	对数组中的每个成员应用用户函数。	3
array_walk_recursive()	对数组中的每个成员递归地应用用户函数。	5
arsort()	对数组进行逆向排序并保持索引关系。	3
asort()	对数组进行排序并保持索引关系。	3
compact()	建立一个数组，包括变量名和它们的值。	4
count()	计算数组中的元素数目或对象中的属性个数。	3
current()	返回数组中的当前元素。	3
each()	返回数组中当前的键／值对并将数组指针向前移动一步。	3
end()	将数组的内部指针指向最后一个元素。	3
extract()	从数组中将变量导入到当前的符号表。	3
in_array()	检查数组中是否存在指定的值。	4
key()	从关联数组中取得键名。	3
krsort()	对数组按照键名逆向排序。	3
ksort()	对数组按照键名排序。	3
list()	把数组中的值赋给一些变量。	3
natcasesort()	用“自然排序”算法对数组进行不区分大小写字母的排序。	4
natsort()	用“自然排序”算法对数组排序。	4
next()	将数组中的内部指针向前移动一位。	3
pos()	current() 的别名。	3
prev()	将数组的内部指针倒回一位。	3
range()	建立一个包含指定范围的元素的数组。	3
reset()	将数组的内部指针指向第一个元素。	3
rsort()	对数组逆向排序。	3
shuffle()	把数组中的元素按随机顺序重新排列。	3
sizeof()	count() 的别名。	3
sort()	对数组排序。	3
uasort()	使用用户自定义的比较函数对数组中的值进行排序并保持索引关联。	3

uksort()	使用用户自定义的比较函数对数组中的键名进行排序。	3
usort()	使用用户自定义的比较函数对数组中的值进行排序。	3

PHP Array 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
CASE_LOWER	用在 array_change_key_case() 中将数组键名转换成小写字母。	
CASE_UPPER	用在 array_change_key_case() 中将数组键名转换成大写字母。	
SORT_ASC	用在 array_multisort() 函数中，使其升序排列。	
SORT_DESC	用在 array_multisort() 函数中，使其降序排列。	
SORT_REGULAR	用于对对象进行通常比较。	
SORT_NUMERIC	用于对对象进行数值比较。	
SORT_STRING	用于对对象进行字符串比较。	
SORT_LOCALE_STRING	基于当前区域来对对象进行字符串比较。	4
COUNT_NORMAL		
COUNT_RECURSIVE		
EXTR_OVERWRITE		
EXTR_SKIP		
EXTR_PREFIX_SAME		
EXTR_PREFIX_ALL		
EXTR_PREFIX_INVALID		
EXTR_PREFIX_IF_EXISTS		
EXTR_IF_EXISTS		
EXTR_REFS		

PHP array()

定义和用法

array() 创建数组，带有键和值。如果在规定数组时省略了键，则生成一个整数键，这个 key 从 0 开始，然后以 1 进行递增。

要用 array() 创建一个关联数组，可使用 => 来分隔键和值。

要创建一个空数组，则不传递参数给 array()：

```
$new = array();
```

注意：array() 实际上是一种语言结构 (language construct)，通常用来定义直接量数组，但它的用法和函数的用法很相似，所以我们把它也列到手册中。

语法

```
array(key => value)
```

参数	描述
key	可选。规定 key，类型是数值或字符串。如果未设置，则生成整数类型的 key。
value	必需。规定值。

例子 1

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
print_r($a);
?>
```

输出：

```
Array ( [a] => Dog [b] => Cat [c] => Horse )
```

例子 2

```
<?php
$a=array("Dog","Cat","Horse");
print_r($a);
?>
```

输出：

```
Array ( [0] => Dog [1] => Cat [2] => Horse )
```

PHP array_change_key_case() 函数

定义和用法

array_change_key_case() 函数将数组的所有的 KEY 都转换为大写或小写。

数组的数字索引不发生变化。如果未提供可选参数（即第二个参数），则默认转换为小写字母。

语法

```
array_change_key_case(array,case)
```

参数	描述
array	必需。规定要使用的数组。
case	可选。可能的值： `CASE_LOWER` - 默认值。以小写字母返回数组的键。 `CASE_UPPER` - 以大写字母返回数组的键。

提示和注释：

注释：如果在运行该函数时两个或多个键相同，则最后的元素会覆盖其他元素（参见例子 2）。

例子 1

```
<?php
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
print_r(array_change_key_case($a,CASE_UPPER));
?>
```

输出：

```
Array ( [A] => Cat [B] => Dog [C] => Horse )
```

例子 2

```
<?php
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse","B"=>"Bird");
print_r(array_change_key_case($a,CASE_UPPER));
?>
```

输出：

```
Array ( [A] => Cat [B] => Bird [C] => Horse )
```

PHP array_chunk() 函数

定义和用法

array_chunk() 函数把一个数组分割为新的数组块。

其中每个数组的单元数目由 size 参数决定。最后一个数组的单元数目可能会少几个。

可选参数 preserve_key 是一个布尔值，它指定新数组的元素是否有和原数组相同的键（用于关联数组），还是从 0 开始的新数字键（用于索引数组）。默认是分配新的键。

语法

```
array_chunk(array,size,preserve_key)
```

参数	描述
array	必需。规定要使用的数组。
size	必需。规定每个新数组包含多少个元素。
preserve_key	可选。可能的值：`true` - 保留原始数组中的键名。 `false` - 默认。每个结果数组使用从零开始的新数组索引。

例子 1

```
<?php
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse","d"=>"Cow");
print_r(array_chunk($a,2));
?>
```

输出：

```
Array (
[0] => Array ( [0] => Cat [1] => Dog )
[1] => Array ( [0] => Horse [1] => Cow )
)
```

例子 2

```
<?php
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse","d"=>"Cow");
print_r(array_chunk($a,2,true));
?>
```

输出：

```
Array (
[0] => Array ( [a] => Cat [b] => Dog )
[1] => Array ( [c] => Horse [d] => Cow )
)
```

PHP array_combine() 函数

定义和用法

array_combine() 函数通过合并两个数组来创建一个新数组，其中的一个数组是键名，另一个数组的值为键值。

如果其中一个数组为空，或者两个数组的元素个数不同，则该函数返回 false。

语法

```
array_combine(array1,array2)
```

参数	描述
array1	必需。规定键名。
array2	必需。规定值。

提示和注释

注释：两个参数必须有相同数目的元素。

例子

```
<?php
$a1=array("a","b","c","d");
$a2=array("Cat","Dog","Horse","Cow");
print_r(array_combine($a1,$a2));
?>
```

输出：

```
Array ( [a] => Cat [b] => Dog [c] => Horse [d] => Cow )
```

PHP array_count_values() 函数

定义和用法

array_count_values() 函数用于统计数组中所有值出现的次数。

本函数返回一个数组，其元素的键名是原数组的值，键值是该值在原数组中出现的次数。

语法

```
array_count_values(array)
```

参数	描述
array	必需。规定输入的数组。

例子

```
<?php
$a=array("Cat","Dog","Horse","Dog");
print_r(array_count_values($a));
?>
```

输出：

```
Array ( [Cat] => 1 [Dog] => 2 [Horse] => 1 )
```

PHP array_diff() 函数

定义和用法

array_diff() 函数返回两个数组的差集数组。该数组包括了所有在被比较的数组中，但是不在任何其他参数数组中的键值。

在返回的数组中，键名保持不变。

语法

```
array_diff(array1,array2,array3...)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。

提示和注释

提示：可用一个或任意多个数组与第一个数组进行比较。

注释：仅有值用于比较。

例子

```
<?php
$a1=array(0=>"Cat",1=>"Dog",2=>"Horse");
$a2=array(3=>"Horse",4=>"Dog",5=>"Fish");
print_r(array_diff($a1,$a2));
?>
```

输出：

```
Array ( [0] => Cat )
```

PHP array_diff_assoc() 函数

定义和用法

array_diff_assoc() 函数返回两个数组的差集数组。该数组包括了所有在被比较的数组中，但是不在任何其他参数数组中的键和值。

和 array_diff() 函数 不同，本函数要求键名和键值都进行比较。返回的数组中键名保持不变。

语法

```
array_diff_assoc(array1,array2,array3...)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。

提示和注释

提示：可用一个或任意多个数组与第一个数组进行比较。

注释：键和值都用于比较。

例子

```
<?php
$a1=array(0=>"Cat",1=>"Dog";,2=>"Horse");
$a2=array(0=>"Rat",1=>"Horse";,2=>"Dog");
$a3=array(0=>"Horse",1=>"Dog",2=>"Cat");
print_r(array_diff_assoc($a1,$a2,$a3));
?>
```

输出：

```
Array ( [0] => Cat [2] => Horse )
```

PHP array_diff_key() 函数

定义和用法

array_diff_key() 函数返回一个数组，该数组包括了所有在被比较的数组中，但是不在任何其他参数数组中的键。

语法

```
array_diff_key(array1,array2,array3...)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。

提示和注释

提示：可用一个或任意多个数组与第一个数组进行比较。

注释：仅仅键名用于比较。

例子

```
<?php
$a1=array(0=>"Cat",1=>"Dog",2=>"Horse");
$a2=array(2=>"Bird",3=>"Rat",4=>"Fish");
$a3=array(5=>"Horse",6=>"Dog",7=>"Bird");
print_r(array_diff_key($a1,$a2,$a3));
?>
```

输出：

```
Array ( [0] => Cat [1] => Dog )
```

PHP array_diff_uassoc() 函数

定义和用法

array*diff_uassoc()* 函数使用用户自定义的回调函数 *(callback)* 做索引检查来计算两个或多个数组的差集。返回一个数组，该数组包括了在 _*array1* 中但是不在任何其他参数数组中的值。

注意，与 array_diff() 函数 不同的是，键名也要进行比较。

参数 function 是用户自定义的用来比较两个数组的函数，该函数必须带有两个参数 - 即两个要进行对比的键名。因此与函数 array_diff_assoc() 的行为正好相反，后者是用内部函数进行比较的。

返回的数组中键名保持不变。

语法

```
array_diff_uassoc(array1,array2,array3...,function)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。
function	必需。用户自定义函数的名称。

例子 1

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
if ($v1>$v2)
    {
    return 1;
    }
else
    {
    return -1;
    }
}
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse");
$a2=array(3=>"Dog",1=>"Cat",5=>"Horse");
print_r(array_diff_uassoc($a1,$a2,"myfunction"));
?>
```

输出：

```
Array ( [0] => Dog [2] => Horse )
```

例子 2

如何为该函数分配多个数组：

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
if ($v1>$v2)
    {
    return 1;
    }
else
    {
    return -1;
    }
}
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse");
$a2=array(3=>"Dog",1=>"Cat",5=>"Horse");
$a3=array(6=>"Bird",0=>"Dog",5=>"Horse");
print_r(array_diff_uassoc($a1,$a2,$a3,"myfunction"));
?>
```

输出：

```
Array ( [2] => Horse )
```

PHP array_diff_ukey() 函数

定义和用法

array_diff_ukey() 返回一个数组，该数组包括了所有出现在 array1 中但是未出现在任何其它参数数组中的键名的值。注意关联关系保留不变。与 array_diff() 不同的是，比较是根据键名而不是值来进行的。

此比较是通过用户提供的回调函数来进行的。如果认为第一个参数小于，等于，或大于第二个参数时必须分别返回一个小于零，等于零，或大于零的整数。

语法

```
array_diff_ukey(array1,array2,array3...,function)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。
function	必需。用户自定义函数的名称。

例子 1

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
if ($v1>$v2)
    {
    return 1;
    }
else
    {
    return -1;
    }
}
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse");
$a2=array(3=>"Rat",1=>"Bird",5=>"Monkey");
print_r(array_diff_ukey($a1,$a2,"myfunction"));
?>
```

输出：

```
Array ( [0] => Dog [2] => Horse )
```

例子 2

如何为该函数分配多个数组：

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
if ($v1>$v2)
    {
    return 1;
    }
else
    {
    return -1;
    }
}
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse");
$a2=array(3=>"Rat",1=>"Bird",5=>"Monkey");
$a3=array(6=>"Dog",7=>"Donkey",0=>"Horse");
print_r(array_diff_ukey($a1,$a2,$a3,"myfunction"));
?>
```

输出：

```
Array ( [2] => Horse )
```

PHP array_fill() 函数

定义和用法

array_fill() 函数用给定的值填充数组，返回的数组有 number 个元素，值为 value。返回的数组使用数字索引，从 start 位置开始并递增。如果 number 为 0 或小于 0，就会出错。

语法

```
array_fill(start,number,value)
```

参数	描述
start	必需。数值，规定键的起始索引。
number	必需。数值，规定填充的数量，其值必须大于 0。
value	必需。规定要插入的值。

例子

```
<?php
$a=array_fill(2,3,"Dog");
print_r($a);
?>
```

输出：

```
Array ( [2] => Dog [3] => Dog [4] => Dog )
```

PHP array_filter() 函数

定义和用法

array_filter() 函数用回调函数过滤数组中的元素，如果自定义过滤函数返回 true，则被操作的数组的当前值就会被包含在返回的结果数组中， 并将结果组成一个新的数组。如果原数组是一个关联数组，键名保持不变。

语法

```
array_filter(array,function)
```

参数	描述
array	必需。规定输入的数组。
function	必需。自定义函数的名称。

例子

```
<?php
function myfunction($v)
{
if ($v==="Horse")
    {
    return true;
    }
return false;
}
$a=array(0=>"Dog",1=>"Cat",2=>"Horse");
print_r(array_filter($a,"myfunction"));
?>
```

输出：

```
Array ( [2] => Horse )
```

PHP array_flip() 函数

定义和用法

array_flip() 函数返回一个反转后的数组，如果同一值出现了多次，则最后一个键名将作为它的值，所有其他的键名都将丢失。

如果原数组中的值的数据类型不是字符串或整数，函数将报错。

语法

```
array_flip(array)
```

参数	描述
array	必需。规定输入的数组。

例子

```
<?php
$a=array(0=>"Dog",1=>"Cat",2=>"Horse");print_r(array_flip($a));
?>
```

输出：

```
Array ( [Dog] => 0 [Cat] => 1 [Horse] => 2 )
```

PHP array_intersect() 函数

定义和用法

array_intersect() 函数返回两个或多个数组的交集数组。

结果数组包含了所有在被比较数组中，也同时出现在所有其他参数数组中的值，键名保留不变。

注释：仅有值用于比较。

语法

```
array_intersect(array1,array2,array3...)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。

例子

```
<?php
$a1=array(0=>"Cat",1=>"Dog",2=>"Horse");
$a2=array(3=>"Horse",4=>"Dog",5=>"Fish");
print_r(array_intersect($a1,$a2));
?>
```

输出：

```
Array ( [1] => Dog [2] => Horse )
```

PHP array_intersect_assoc() 函数

定义和用法

array_intersect_assoc() 函数返回两个或多个数组的交集数组。

与 array_intersect() 函数 不同的是，本函数除了比较键值，还比较键名。返回的数组中元素的键名保持不变。

语法

```
array_intersect_assoc(array1,array2,array3...)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。

例子 1

```
<?php
$a1=array(0=>"Cat",1=>"Dog",2=>"Horse");
$a2=array(3=>"Horse",1=>"Dog",0=>"Cat");
print_r(array_intersect_assoc($a1,$a2));
?>
```

输出：

```
Array ( [0] => Cat [1] => Dog )
```

例子 2

```
<?php
$a1=array(0=>"Cat",1=>"Dog",2=>"Horse");
$a2=array(3=>"Horse",1=>"Dog",5=>"Fish");
$a3=array(6=>"Cow",1=>"Dog",8=>"Fish");
print_r(array_intersect_assoc($a1,$a2,$a3));
?>
```

输出：

```
Array ( [1] => Dog )
```

PHP array_intersect_key() 函数

定义和用法

array_intersect_key() 函数使用键名比较计算数组的交集。

array_intersect_key() 返回一个数组，该数组包含了所有出现在被比较的数组中并同时出现在所有其它参数数组中的键名的值。

注释：仅有键名用于比较。

语法

```
array_intersect_key(array1,array2,array3...)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。

例子 1

```
<?php
$a1=array(0=>"Cat",1=>"Dog",2=>"Horse");
$a2=array(2=>"Bird",0=>"Cat",4=>"Fish");
print_r(array_intersect_key($a1,$a2));
?>
```

输出：

```
Array ( [0] => Cat [2] => Horse )
```

例子 2

```
<?php
$a1=array(0=>"Cat",1=>"Dog",2=>"Horse");
$a2=array(2=>"Bird",3=>"Rat",4=>"Fish");
$a3=array(2=>"Dog",6=>"Cow",7=>"Bird");
print_r(array_intersect_key($a1,$a2,$a3));
?>
```

输出：

```
Array ( [2] => Horse )
```

PHP array_intersect_uassoc() 函数

定义和用法

array_intersect_uassoc() 函数使用用户自定义的回调函数计算数组的交集，用回调函数比较索引。

array*intersect_uassoc()* 返回一个数组，该数组包含了所有在 _*array1* 中也同时出现在所有其它参数数组中的值。返回的数组中键名保持不变。

注意，与 array_intersect() 不同的是除了比较键值，还要比较键名。

此比较是通过用户提供的回调函数来进行的。该函数带有两个参数，即两个要进行对比的键名。如果第一个参数小于第二个参数，则函数要返回一个负数，如果两个参数相等，则要返回 0，如果第一个参数大于第二个参数，则返回一个正数。

语法

```
array_intersect_uassoc(array1,array2,array3...,function)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。
function	用户自定义函数的名称。

例子 1

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
if ($v1>$v2)
    {
    return 1;
    }
else
    {
    return -1;
    }
}
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse");
$a2=array(3=>"Dog",1=>"Cat",5=>"Horse");
print_r(array_intersect_uassoc($a1,$a2,"myfunction"));
?>
```

输出：

```
Array ( [1] => Cat )
```

例子 2

如何为函数分配多个数组：

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
if ($v1>$v2)
    {
    return 1;
    }
else
    {
    return -1;
    }
}
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse");
$a2=array(0=>"Dog",1=>"Cat",5=>"Horse");
$a3=array(6=>"Bird",0=>"Dog",5=>"Horse");
print_r(array_intersect_uassoc($a1,$a2,$a3,"myfunction"));
?>
```

输出：

```
Array ( [0] => Dog )
```

PHP array_intersect_ukey() 函数

定义和用法

array_intersect_ukey() 函数用回调函数比较键名来计算数组的交集。

array_intersect_ukey() 返回一个数组，该数组包含了所有出现在 array1 中并同时出现在所有其它参数数组中的键名的值。

此比较是通过用户提供的回调函数来进行的。该函数带有两个参数，即两个要进行对比的键名。如果第一个参数小于第二个参数，则函数要返回一个负数，如果两个参数相等，则要返回 0，如果第一个参数大于第二个参数，则返回一个正数。

语法

```
array_intersect_uassoc(array1,array2,array3...,function)
```

参数	描述
array1	必需。与其他数组进行比较的第一个数组。
array2	必需。与第一个数组进行比较的数组。
array3	可选。与第一个数组进行比较的数组。可以有多个。
function	用户自定义函数的名称。

例子 1

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
if ($v1>$v2)
    {
    return 1;
    }
else
    {
    return -1;
    }
}
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse");
$a2=array(3=>"Rat",1=>"Bird",5=>"Monkey");
print_r(array_intersect_ukey($a1,$a2,"myfunction"));
?>
```

输出：

```
Array ( [1] => Cat )
```

例子 2

如何为函数分配多个数组：

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
if ($v1>$v2)
    {
    return 1;
    }
else
    {
    return -1;
    }
}
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse");
$a2=array(0=>"Rat",1=>"Bird",5=>"Monkey");
$a3=array(6=>"Dog",7=>"Donkey",0=>"Horse");
print_r(array_intersect_ukey($a1,$a2,$a3,"myfunction"));
?>
```

输出：

```
Array ( [0] => Dog )
```

PHP array_key_exists() 函数

定义和用法

array_key_exists() 函数判断某个数组中是否存在指定的 key，如果该 key 存在，则返回 true，否则返回 false。

语法

```
array_key_exists(key,array)
```

参数	描述
key	必需。规定键名。
array	必需。规定输入的数组。

例子 1

```
<?php
$a=array("a"=>"Dog","b"=>"Cat");
if (array_key_exists("a",$a))
    {
    echo "Key exists!";
    }
else
    {
    echo "Key does not exist!";
    }
?>
```

输出：

```
Key exists!
```

例子 2

```
<?php
$a=array("a"=>"Dog","b"=>"Cat");
if (array_key_exists("c",$a))
    {
    echo "Key exists!";
    }
else
    {
    echo "Key does not exist!";
    }
?>
```

输出：

```
Key does not exist!
```

例子 2

```
<?php
$a=array("Dog",Cat");
if (array_key_exists(0,$a))
    {
    echo "Key exists!";
    }
else
    {
    echo "Key does not exist!";
    }
?>
```

输出：

```
Key exists!
```

PHP array_keys() 函数

定义和用法

array_keys() 函数返回包含数组中所有键名的一个新数组。

如果提供了第二个参数，则只返回键值为该值的键名。

如果 strict 参数指定为 true，则 PHP 会使用全等比较 (===) 来严格检查键值的数据类型。

语法

```
array_keys(array,value)
```

参数	描述
array	必需。规定输入的数组。
value	可选。指定值的索引（键）。
strict	可选。与 value 参数一起使用。可能的值： `true` - 根据类型返回带有指定值的键名。 `false` - 默认值。不依赖类型。

例子 1

```
<?php
$a=array("a"=>"Horse","b"=>"Cat","c"=>"Dog");
print_r(array_keys($a));
?>
```

输出：

```
Array ( [0] => a [1] => b [2] => c )
```

例子 2

使用 value 参数：

```
<?php
$a=array("a"=>"Horse","b"=>"Cat","c"=>"Dog");
print_r(array_keys($a,"Dog"));
?>
```

输出：

```
Array ( [0] => c)
```

例子 3

使用 strict 参数 (false)：

```
<?php
$a=array(10,20,30,"10");
print_r(array_keys($a,"10",false));
?>
```

输出：

```
Array ( [0] => 0 [1] => 3 )
```

例子 4

使用 strict 参数 (true)：

```
<?php
$a=array(10,20,30,"10");
print_r(array_keys($a,"10",true));
?>
```

输出：

```
Array ( [0] => 3)
```

PHP array_map() 函数

定义和用法

array_map() 函数返回用户自定义函数作用后的数组。回调函数接受的参数数目应该和传递给 array_map() 函数的数组数目一致。

语法

```
array_map(function,array1,array2,array3...)
```

参数	描述
function	必需。用户自定义函数的名称，或者是 null。
array1	必需。规定数组。
array2	可选。规定数组。
array3	可选。规定数组。

例子 1

```
<?php
function myfunction($v)
{
if ($v==="Dog")
    {
    return "Fido";
    }
return $v;
}
$a=array("Horse","Dog","Cat");
print_r(array_map("myfunction",$a));
?>
```

输出：

```
Array ( [0] => Horse [1] => Fido [2] => Cat )
```

例子 2

使用多个参数：

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return "same";
    }
return "different";
}
$a1=array("Horse","Dog","Cat");
$a2=array("Cow","Dog","Rat");
print_r(array_map("myfunction",$a1,$a2));
?>
```

输出：

```
Array ( [0] => different [1] => same [2] => different )
```

例子 3

请看当自定义函数名设置为 null 时的情况：

```
<?php
$a1=array("Dog","Cat");
$a2=array("Puppy","Kitten");
print_r(array_map(null,$a1,$a2));
?>
```

输出：

```
Array (
[0] => Array ( [0] => Dog [1] => Puppy )
[1] => Array ( [0] => Cat [1] => Kitten )
)
```

PHP array_merge() 函数

定义和用法

array_merge() 函数把两个或多个数组合并为一个数组。

如果键名有重复，该键的键值为最后一个键名对应的值（后面的覆盖前面的）。如果数组是数字索引的，则键名会以连续方式重新索引。

注释：如果仅仅向 array_merge() 函数输入了一个数组，且键名是整数，则该函数将返回带有整数键名的新数组，其键名以 0 开始进行重新索引。（参见例子 2）

语法

```
array_merge(array1,array2,array3...)
```

参数	描述
array1	必需。输入的第一个数组。
array2	必需。输入的第二个数组。
array3	可选。可指定的多个输入数组。

例子 1

```
<?php
$a1=array("a"=>"Horse","b"=>"Dog");
$a2=array("c"=>"Cow","b"=>"Cat");
print_r(array_merge($a1,$a2));
?>
```

输出：

```
Array ( [a] => Horse [b] => Cat [c] => Cow )
```

例子 2

仅使用一个数组参数：

```
<?php
$a=array(3=>"Horse",4=>"Dog");
print_r(array_merge($a));
?>
```

输出：

```
Array ( [0] => Horse [1] => Dog )
```

PHP array_merge_recursive() 函数

定义和用法

array_merge_recursive() 函数与 array_merge() 函数 一样，将一个或多个数组的元素的合并起来，一个数组中的值附加在前一个数组的后面。并返回作为结果的数组。

但是，与 array_merge() 不同的是，当有重复的键名时，值不会被覆盖，而是将多个相同键名的值递归组成一个数组。（参见例子 1）

语法

```
array_merge_recursive(array1,array2,array3...)
```

参数	描述
array1	必需。输入的第一个数组。
array2	必需。输入的第二个数组。
array3	可选。可指定的多个输入数组。

提示和注释

注释：当向 array_merge_recursive() 函数仅仅输入一个数组时，结果与 array_merge() 相同。

例子 1

```
<?php
$a1=array("a"=>"Horse","b"=>"Dog");
$a2=array("c"=>"Cow","b"=>"Cat");
print_r(array_merge_recursive($a1,$a2));
?>
```

输出：

```
Array (
[a] => Horse
[b] => Array ( [0] => Dog [1] => Cat )
[c] => Cow
)
```

PHP array_multisort() 函数

定义和用法

array_multisort() 函数对多个数组或多维数组进行排序。

参数中的数组被当成一个表的列并以行来进行排序 - 这类似 SQL 的 ORDER BY 子句的功能。第一个数组是要排序的主要数组。数组中的行（值）比较为相同的话，就会按照下一个输入数组中相应值的大小进行排序，依此类推。

第一个参数是数组，随后的每一个参数可能是数组，也可能是下面的排序顺序标志（排序标志用于更改默认的排列顺序）之一：

- SORT_ASC - 默认，按升序排列。(A-Z)
- SORT_DESC - 按降序排列。(Z-A)

随后可以指定排序的类型：

- SORT_REGULAR - 默认。将每一项按常规顺序排列。
- SORT_NUMERIC - 将每一项按数字顺序排列。
- SORT_STRING - 将每一项按字母顺序排列。

语法

```
array_multisort(array1,sorting order,sorting type,array2,array3...)
```

参数	描述
array1	必需。规定输入的数组。
sorting order	可选。规定排列顺序。可能的值是 SORT_ASC 和 SORT_DESC。
sorting type	可选。规定排序类型。可能的值是SORT_REGULAR、SORT_NUMERIC和SORT_STRING。
array2	可选。规定输入的数组。
array3	可选。规定输入的数组。

提示和注释

注释：字符串键名将被保留，但是数字键将被重新索引，从 0 开始，并以 1 递增。

注释：您可以在每个数组后设置排序顺序和排序类型。如果没有设置，每个数组参数会使用默认值。

例子 1

```
<?php
$a1=array("Dog","Cat");
$a2=array("Fido","Missy");
array_multisort($a1,$a2);
print_r($a1);
print_r($a2);
?>
```

输出：

```
Array ( [0] => Cat [1] => Dog )
Array ( [0] => Missy [1] => Fido )
```

例子 2

当两个值相同时如何排序：

```
<?php
$a1=array("Dog","Dog","Cat");
$a2=array("Pluto","Fido","Missy");
array_multisort($a1,$a2);
print_r($a1);
print_r($a2);
?>
```

输出：

```
Array ( [0] => Cat [1] => Dog [2] => Dog )
Array ( [0] => Missy [1] => Fido [2] => Pluto )
```

例子 3

带有排序参数：

```
<?php
$a1=array("Dog","Dog","Cat");
$a2=array("Pluto","Fido","Missy");
array_multisort($a1,SORT_ASC,$a2,SORT_DESC);
print_r($a1);
print_r($a2);
?>
```

输出：

```
Array ( [0] => Cat [1] => Dog [2] => Dog )
Array ( [0] => Missy [1] => Pluto [2] => Fido )
```

PHP array_pad() 函数

定义和用法

array_pad() 函数向一个数组插入带有指定值的指定数量的元素。

语法

```
array_pad(array,size,value)
```

参数	描述
array	必需。规定数组。
size	必需。指定的长度。整数则填补到右侧，负数则填补到左侧。
value	必需。用来填补的值。

提示和注释

提示：如何设置了负的长度值，该函数会在原始数组之前插入新的元素。（参见例子 2）

注释：如果 size 参数小于原始数组的长度，该函数不会删除任何元素。

例子 1

```
<?php
$a=array("Dog","Cat");
print_r(array_pad($a,5,0));
?>
```

输出：

```
Array ( [0] => Dog [1] => Cat [2] => 0 [3] => 0 [4] => 0 )
```

例子 2

带有负的 size 参数：

```
<?php
$a=array("Dog","Cat");
print_r(array_pad($a,-5,0));
?>
```

输出：

```
Array ( [0] => 0 [1] => 0 [2] => 0 [3] => Dog [4] => Cat )
```

PHP array_pop() 函数

定义和用法

array_pop() 函数删除数组中的最后一个元素。

语法

```
array_pop(array)
```

参数	描述
array	必需。规定输入的数组参数。

例子

```
<?php
$a=array("Dog","Cat","Horse");
array_pop($a);
print_r($a);
?>
```

输出：

```
Array ( [0] => Dog [1] => Cat )
```

PHP array_product() 函数

定义和用法

array_product() 函数计算并返回数组中所有值的乘积。

语法

```
array_product(array)
```

参数	描述
array	必需。规定输入的数组参数。

例子

```
<?php
$a=array(5,5);
echo(array_product($a));
?>
```

输出：

```
25
```

PHP array_push() 函数

定义和用法

array_push() 函数向第一个参数的数组尾部添加一个或多个元素（入栈），然后返回新数组的长度。

该函数等于多次调用 $array[] = $value。

语法

```
array_push(array,value1,value2...)
```

参数	描述
array	必需。规定一个数组。
value1	必需。规定要添加的值。
value2	可选。规定要添加的值。

提示和注释

注释：即使数组中有字符串键名，您添加的元素也始终是数字键。（参见例子 2）

注释：如果用 array*push()* 来给数组增加一个单元，还不如用 $array[] =_，因为这样没有调用函数的额外负担。

注释：如果第一个参数不是数组，array_push() 将发出一条警告。这和 $var[] 的行为不同，后者会新建一个数组。

例子 1

```
<?php
$a=array("Dog","Cat");
array_push($a,"Horse","Bird");
print_r($a);
?>
```

输出：

```
Array ( [0] => Dog [1] => Cat [2] => Horse [3] => Bird )
```

例子 2

带有字符串键的数组：

```
<?php
$a=array("a"=>"Dog","b"=>"Cat");
array_push($a,"Horse","Bird");
print_r($a);
?>
```

输出：

```
Array ( [a] => Dog [b] => Cat [0] => Horse [1] => Bird )
```

PHP array_rand() 函数

定义和用法

array_rand() 函数从数组中随机选出一个或多个元素，并返回。

第二个参数用来确定要选出几个元素。如果选出的元素不止一个，则返回包含随机键名的数组，否则返回该元素的键名。

注释：自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 函数给随机数发生器播种，现已被自动完成。

语法

```
array_rand(array,number)
```

参数	描述
array	必需。规定输入的数组参数。
number	可选。默认是 1。规定返回多少个随机的元素。

例子 1

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
print_r(array_rand($a,1));
?>
```

输出：

```
b
```

例子 2

带有字符串键的数组：

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
print_r(array_rand($a,2));
?>
```

输出：

```
Array ( [0] => c [1] => b )
```

PHP array_reduce() 函数

定义和用法

array_reduce() 函数用回调函数迭代地将数组简化为单一的值。如果指定第三个参数，则该参数将被当成是数组中的第一个值来处理，或者如果数组为空的话就作为最终返回值。

语法

```
array_reduce(array,function,initial)
```

参数	描述
array	必需。规定输入的数组。
function	必需。规定自定义回调函数的名称。
initial	可选。如果规定了该参数，该参数将作为数组中的第一个值来处理。

例子 1

```
<?php
function myfunction($v1,$v2)
{
return $v1 . "-" . $v2;
}
$a=array("Dog","Cat","Horse");
print_r(array_reduce($a,"myfunction"));
?>
```

输出：

```
-Dog-Cat-Horse
```

例子 2

带有 initial 参数：

```
<?php
function myfunction($v1,$v2)
{
return $v1 . "-" . $v2;
}
$a=array("Dog","Cat","Horse");
print_r(array_reduce($a,"myfunction",5));
?>
```

输出：

```
5-Dog-Cat-Horse
```

例子 3

返回总和：

```
<?php
function myfunction($v1,$v2)
{
return $v1+$v2;
}
$a=array(10,15,20);
print_r(array_reduce($a,"myfunction",5));
?>
```

输出：

```
50
```

PHP array_reverse() 函数

定义和用法

array_reverse() 函数将原数组中的元素顺序翻转，创建新的数组并返回。如果第二个参数指定为 true，则元素的键名保持不变，否则键名将丢失。

语法

```
array_reverse(array,preserve)
```

参数	描述
array	必需。规定数组。
preserve	可选。规定是否保留原始数组的键名。可能的值：true 和 false。这个参数是 PHP 4.0.3 中新加的。

例子

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
print_r(array_reverse($a));
?>
```

输出：

```
Array ( [c] => Horse [b] => Cat [a] => Dog )
```

PHP array_search() 函数

定义和用法

array_search() 函数与 in_array() 一样，在数组中查找一个键值。如果找到了该值，匹配元素的键名会被返回。如果没找到，则返回 false。

在 PHP 4.2.0 之前，函数在失败时返回 null 而不是 false。

如果第三个参数 strict 被指定为 true，则只有在数据类型和值都一致时才返回相应元素的键名。

语法

```
array_search(value,array,strict)
```

参数	描述
value	必需。规定在数组中搜索的值。
array	必需。被搜索的数组。
strict	可选。可能的值：`true` `false` - 默认 如果值设置为 true，还将在数组中检查给定值的类型。（参见例子 2）

例子 1

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
echo array_search("Dog",$a);
?>
```

输出：

```
a
```

例子 2

```
<?php
$a=array("a"=>"5","b"=>5,"c"=>"5");
echo array_search(5,$a,true);
?>
```

输出：

```
b
```

PHP array_shift() 函数

定义和用法

array_shift() 函数删除数组中的第一个元素，并返回被删除元素的值。

注释：如果键是数字的，所有元素都将获得新的键，从 0 开始，并以 1 递增。（参见例子 2）。

语法

```
array_shift(array)
```

参数	描述
array	必需。规定输入的数组。

例子 1

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
echo array_shift($a);
print_r ($a);
?>
```

输出：

```
Dog
Array ( [b] => Cat [c] => Horse )
```

例子 2

带有数字键：

```
<?php
$a=array(0=>"Dog",1=>"Cat",2=>"Horse");
echo array_shift($a);
print_r ($a);
?>
```

输出：

```
Dog
Array ( [0] => Cat [1] => Horse )
```

PHP array_slice() 函数

定义和用法

array_slice() 函数在数组中根据条件取出一段值，并返回。

注释：如果数组有字符串键，所返回的数组将保留键名。（参见例子 4）

语法

```
array_slice(_array_,_offset_,_length_,_preserve_)
```

参数	描述
array	必需。规定输入的数组。
offset	必需。数值。规定取出元素的开始位置。如果是正数，则从前往后开始取，如果是负值，从后向前取 *offset* 绝对值。
length	可选。数值。规定被返回数组的长度。如果 *length* 为正，则返回该数量的元素。如果 *length* 为负，则序列将终止在距离数组末端这么远的地方。如果省略，则序列将从 *offset* 开始直到 *array* 的末端。
preserve	可选。可能的值： `true` - 保留键 `false` - 默认 - 重置键

例子 1

```
<?php
$a=array(0=>"Dog",1=>"Cat",2=>"Horse",3=>"Bird");
print_r(array_slice($a,1,2));
?>
```

输出：

```
Array ( [0] => Cat [1] => Horse )
```

例子 2

带有负的 offset 参数：

```
<?php
$a=array(0=>"Dog",1=>"Cat",2=>"Horse",3=>"Bird");
print_r(array_slice($a,-2,1));
?>
```

输出：

```
Array ( [0] => Horse )
```

例子 3

preserve 参数设置为 true：

```
<?php
$a=array(0=>"Dog",1=>"Cat",2=>"Horse",3=>"Bird");
print_r(array_slice($a,1,2,true));
?>
```

输出：

```
Array ( [1] => Cat [2] => Horse )
```

例子 4

带有字符串键：

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse","d"=>"Bird");
print_r(array_slice($a,1,2));
?>
```

输出：

```
Array ( [b] => Cat [c] => Horse )
```

PHP array_splice() 函数

定义和用法

array_splice() 函数与 array_slice() 函数类似，选择数组中的一系列元素，但不返回，而是删除它们并用其它值代替。

如果提供了第四个参数，则之前选中的那些元素将被第四个参数指定的数组取代。

最后生成的数组将会返回。

语法

```
array_splice(array,offset,length,array)
```

参数	描述
array	必需。规定数组。
offset	必需。数值。如果 offset 为正，则从输入数组中该值指定的偏移量开始移除。如果 offset 为负，则从输入数组末尾倒数该值指定的偏移量开始移除。
length	可选。数值。如果省略该参数，则移除数组中从 offset 到 结尾的所有部分。如果指定了 length 并且为正值，则移除这么多元素。如果指定了 length 且为负值，则移除从 offset 到数组末尾倒数 length 为止中间所有的元素。
array	被移除的元素由此数组中的元素替代。如果没有移除任何值，则此数组中的元素将插入到指定位置。

提示和注释

提示：如果函数没有删除任何元素 (length=0)，则替代数组将从 start 参数的位置插入。（参见例子 3）

注释：不保留替代数组中的键。

例子 1

```
<?php
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse",3=>"Bird");
$a2=array(0=>"Tiger",1=>"Lion");
array_splice($a1,0,2,$a2);
print_r($a1);
?>
```

输出：

```
Array ( [0] => Tiger [1] => Lion [2] => Horse [3] => Bird )
```

例子 2

与例子 1 相同，但是输出返回的数组：

```
<?php
$a1=array(0=>"Dog",1=>"Cat",2=>"Horse",3=>"Bird");
$a2=array(0=>"Tiger",1=>"Lion");
print_r(array_splice($a1,0,2,$a2));
?>
```

输出：

```
Array ( [0] => Dog [1] => Cat )
```

例子 3

length 参数设置为 0：

```
<?php
$a1=array(0=>"Dog",1=>"Cat");
$a2=array(0=>"Tiger",1=>"Lion");
array_splice($a1,1,0,$a2);
print_r($a1);
?>
```

输出：

```
Array ( [0] => Dog [1] => Tiger [2] => Lion [3] => Cat )
```

PHP array_sum() 函数

定义和用法

array_sum() 函数返回数组中所有值的总和。

如果所有值多是整数，则返回一个整数值。如果其中有一个或多个值是浮点数，则返回浮点数。

PHP 4.2.1 之前的版本修改了传入的数组本身，将其中的字符串值转换成数值（大多数情况下都转换成了零，根据具体制而定）。

语法

```
array_sum(array)
```

参数	描述
array	必需。规定输入的数组。

例子

```
<?php
$a=array(0=>"5",1=>"15",2=>"25");
echo array_sum($a);
?>
```

输出：

```
45
```

PHP array_udiff() 函数

定义和用法

array_udiff() 函数返回一个数组，该数组包括了所有在被比较数组中，但是不在任何其它参数数组中的值，键名保留不变。

array_udiff() 函数与 array_diff() 函数 的行为不同，后者用内部函数进行比较。

数据的比较是用 array*udiff()* 函数的 _*function* 进行的。*function* 函数带有两个将进行比较的参数。如果第一个参数小于第二个参数，则函数返回一个负数，如果两个参数相等，则要返回 0，如果第一个参数大于第二个，则返回一个正数。

语法

```
array_udiff(_array1_,_array2_,_array3_...,_function_)
```

参数	描述
array1	必需。被比较的数组。
array2	必需。用来做比较的数组。
array3	可选。用来做比较的数组，可有多个。
function	必需。自定义的比较回调函数。

注释和提示

注释：该函数只进行键值的比较，不比较键名。如 "a"=>1 和 "b"=>1 这两个元素视作相等的。

注释：array_udiff() 函数仅检查多维数组中的一维。

例子

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
return 1;
}
$a1=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
$a2=array(1=>"Cat",2=>"Dog",3=>"Fish");
print_r(array_udiff($a1,$a2,"myfunction"));
?>
```

输出：

```
Array ( [c] => Horse )
```

PHP array_udiff_assoc() 函数

定义和用法

array*udiff_assoc()* 函数返回 _*array1* 中存在但其它数组中都不存在的部分。

注意与 array_diff() 以及 array_udiff() 不同的是键名也用于比较。同时进行键名和键值的比较。如 "a"=>1 和 "b"=>1 这两个元素是不相等的。

数组数据的比较是用用户提供的回调函数进行的。在此方面和 array_diff_assoc() 的行为正好相反，后者是用内部函数进行比较的。

array*udiff_assoc()* 函数的 _*function* 参数指定的函数用于比较元素是否相等。*function* 函数带有两个将进行比较的参数。如果第一个参数小于第二个参数，则函数返回一个负数，如果两个参数相等，则要返回 0，如果第一个参数大于第二个，则返回一个正数。

语法

```
array_udiff_assoc(array1,array2,array3...,function)
```

参数	描述
array1	必需。被比较的数组。
array2	必需。用来做比较的数组。
array3	可选。用来做比较的数组，可有多个。
function	可选。自定义的比较回调函数。

例子

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
return 1;
}
$a1=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
$a2=array("a"=>"Cat","b"=>"Horse","c"=>"Dog");
print_r(array_udiff_assoc($a1,$a2,"myfunction"));
?>
```

输出：

```
Array ( [b] => Dog [c] => Horse )
```

PHP array_udiff_uassoc() 函数

定义和用法

array*udiff_uassoc()* 函数返回 _*array1* 数组中存在但其它数组中都不存在的部分。返回的数组中键名保持不变。

注意与 array_diff() 以及 array_udiff() 不同的是键名也用于比较。同时进行键名和键值的比较，如 "a"=>1 和 "b"=>1 这两个元素是不相等的。

对键名（索引）的检查也是由回调函数 *function1* 进行的。这和 array_udiff_assoc() 的行为不同，后者是用内部函数比较索引的。

数组数据的比较是使用用户提供的回调函数 *function2* 进行的。在此方面和 array_diff_assoc() 的行为正好相反，后者是用内部函数进行比较的。

这两个函数都带有两个将进行比较的参数。如果第一个参数小于第二个参数，则函数返回一个负数，如果两个参数相等，则要返回 0，如果第一个参数大于第二个，则返回一个正数。

语法

```
array_udiff_uassoc(array1,array2,array3...,function1,function2)
```

参数	描述
array1	必需。被比较的数组。
array2	必需。用来做比较的数组。
array3	可选。用来做比较的数组，可有多个。
function1	必需。比较键名的自定义函数。
function2	必需。比较值的自定义函数。

注释：*function1* 指定的函数用于比较键名是否相等。*function2* 指定的函数用于比较键值是否相等。

例子

```
<?php
function myfunction_key($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
return 1;
}
function myfunction_value($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
return 1;
}
$a1=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
$a2=array("a"=>"Cat","b"=>"Dog","c"=>"Fish");
print_r(array_udiff_uassoc($a1,$a2,"myfunction_key","myfunction_value"));
?>
```

输出：

```
Array ( [c] => Horse )
```

PHP array_uintersect() 函数

定义和用法

array_uintersect() 函数计算数组的交集，用回调函数比较数据。

array*uintersect()* 返回一个数组，该数组包含了所有在 _*array1* 中也同时出现在所有其它参数数组中的值。数据（键值）比较是用回调函数进行的。

语法

```
array_uintersect(array1,array2,array3...,function)
```

参数	描述
array1	必需。被比较的数组。
array2	必需。用来做比较的数组。
array3	可选。用来做比较的数组，可有多个。
function	必需。自定义函数的名称。

说明

使用用户自定义的回调函数 *function* 来计算两个或多个数组的交集（即在 *array1* 中存在同时也在其它任何数组中存在的所有数组元素），并返回结果数组。

只进行键值的比较，不比较键名，如 "a"=>1 和 "b"=>1 这两个元素视作相等的。

function 参数指定的函数用于比较元素是否相等。function 函数带有两个将进行比较的参数。如果第一个参数小于第二个参数，则函数返回一个负数，如果两个参数相等，则要返回 0，如果第一个参数大于第二个，则返回一个正数。

返回的数组中键名保持不变。

例子

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
  {
  return 0;
  }
if ($v1 > $v2) return 1;
  {
  return -1;
  }
return 1;
}
$a1=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
$a2=array(1=>"Cat",2=>"Dog",3=>"Fish");
print_r(array_uintersect($a1,$a2,"myfunction"));
?>
```

输出：

```
Array ( [a] => Cat [b] => Dog )
```

PHP array_uintersect_assoc() 函数

定义和用法

array_uintersect_assoc() 函数带索引检查计算数组的交集，用回调函数比较数据。

array*uintersect_assoc()* 返回一个数组，该数组包含了所有在 _*array1* 中也同时出现在所有其它参数数组中的值。

注意，与 array_uintersect() 不同的是键名也要比较。数据（键值）是用回调函数比较的。

语法

```
array_uintersect_assoc(array1,array2,array3...,function)
```

参数	描述
array1	必需。被比较的数组。
array2	必需。用来做比较的数组。
array3	可选。用来做比较的数组，可有多个。
function	必需。自定义函数的名称。

说明

使用用户自定义的回调函数 *function* 来计算两个或多个数组的交集（即在 *array1* 中存在，同时也在其它任何数组中存在的所有数组元素），并返回结果数组。

同时进行键名和键值的比较，如 "a"=>1 和 "b"=>1 这两个元素是不相等的。

function 参数指定的函数用于比较元素是否相等。*function* 函数带有两个将进行比较的参数。如果第一个参数小于第二个参数，则函数返回一个负数，如果两个参数相等，则要返回 0，如果第一个参数大于第二个，则返回一个正数。

返回的数组中键名保持不变。

例子

```
<?php
function myfunction($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
return 1;
}
$a1=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
$a2=array("a"=>"Cat","b"=>"Horse","c"=>"Dog");
print_r(array_uintersect_assoc($a1,$a2,"myfunction"));
?>
```

输出：

```
Array ( [a] => Cat )
```

PHP array_uintersect_uassoc() 函数

定义和用法

array_uintersect_uassoc 函数带索引检查计算数组的交集，用回调函数来比较数据和索引。

array*uintersect_uassoc()* 返回一个数组，该数组包含了所有在 _*array1* 中也同时出现在所有其它参数数组中的值。

注意，与 array_uintersect() 不同的是键名也要比较。键值和键名（索引）都是用回调函数比较的。

语法

```
array_uintersect_uassoc(array1,array2,array3...,function1,function2)
```

参数	描述
array1	必需。被比较的数组。
array2	必需。用来做比较的数组。
array3	可选。用来做比较的数组，可有多个。
function1	必需。比较键名的自定义函数。
function2	必需。比较键值的自定义函数。

说明

使用用户自定义的回调函数 *function1* 和 *function2* 来计算两个或多个数组的交集（即在 *array1* 中存在，同时也在其它任何数组中存在的所有数组元素），并返回结果数组。

同时进行键名和键值的比较，如 "a"=>1 和 "b"=>1 这两个元素是不相等的。

function1 指定的函数用于比较键名是否相等。*function2* 指定的函数用于比较键值是否相等。这两个函数都带有两个将进行比较的参数。如果第一个参数小于第二个参数，则函数返回一个负数，如果两个参数相等，则要返回 0，如果第一个参数大于第二个，则返回一个正数。

返回的数组中键名保持不变。

例子

```
<?php
function myfunction_key($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
return 1;
}

function myfunction_value($v1,$v2)
{
if ($v1===$v2)
    {
    return 0;
    }
return 1;
}
$a1=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
$a2=array("a"=>"Cat","b"=>"Dog","c"=>"Dog");
print_r(array_uintersect_uassoc($a1,$a2,"myfunction_key","myfunction_value"));
?>
```

输出：

```
Array ( [a] => Cat [b] => Dog )
```

PHP array_unique() 函数

定义和用法

array_unique() 函数移除数组中的重复的值，并返回结果数组。

当几个数组元素的值相等时，只保留第一个元素，其他的元素被删除。

返回的数组中键名不变。

语法

```
array_unique(array)
```

参数	描述
array	必需。规定输入的数组。

说明

array*unique()* 先将值作为字符串排序，然后对每个值只保留第一个遇到的键名，接着忽略所有后面的键名。这并不意味着在未排序的 _*array* 中同一个值的第一个出现的键名会被保留。

提示和注释

注释：被返回的数组将保持第一个数组元素的键类型。

例子

```
<?php
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Cat");
print_r(array_unique($a));
?>
```

输出：

```
Array ( [a] => Cat [b] => Dog )
```

PHP array_unshift() 函数

定义和用法

array_unshift() 函数在数组开头插入一个或多个元素。

被加上的元素作为一个整体添加，这些元素在数组中的顺序和在参数中的顺序一样。

该函数会返回数组中元素的个数。

语法

```
array_unshift(array,value1,value2,value3...)
```

参数	描述
array	必需。规定输入的数组。
value1	必需。规定插入的值。
value2	可选。规定插入的值。
value3	可选。规定插入的值。

提示和注释

注释：所有的数值键名将修改为从零开始重新计数，所有的字符串键名保持不变。

例子 1

```
<?php
$a=array("a"=>"Cat","b"=>"Dog");
array_unshift($a,"Horse");
print_r($a);
?>
```

输出：

```
Array ( [0] => Horse [a] => Cat [b] => Dog )
```

例子 2

返回键值：

```
<?php
$a=array("a"=>"Cat","b"=>"Dog");
print_r(array_unshift($a,"Horse"));
?>
```

输出：

```
3
```

例子 3

数组带有数值键：

```
<?php
$a=array(0=>"Cat",1=>"Dog");
array_unshift($a,"Horse");
print_r($a);
?>
```

输出：

```
Array ( [0] => Horse [1] => Cat [2] => Dog )
```

PHP array_values() 函数

定义和用法

array_values() 函数返回一个包含给定数组中所有键值的数组，但不保留键名。

语法

```
array_values(array)
```

参数	描述
array	必需。规定给定的数组。

提示和注释

提示：被返回的数组将使用数值键，从 0 开始且以 1 递增。

例子

```
<?php
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
print_r(array_values($a));
?>
```

输出：

```
Array ( [0] => Cat [1] => Dog [2] => Horse )
```

PHP array_walk() 函数

定义和用法

array_walk() 函数对数组中的每个元素应用回调函数。如果成功则返回 TRUE，否则返回 FALSE。

典型情况下 *function* 接受两个参数。*array* 参数的值作为第一个，键名作为第二个。如果提供了可选参数 *userdata* ，将被作为第三个参数传递给回调函数。

如果 *function* 函数需要的参数比给出的多，则每次 array*walk()* 调用 _*function* 时都会产生一个 E_WARNING 级的错误。这些警告可以通过在 array_walk() 调用前加上 PHP 的错误操作符 @ 来抑制，或者用 error_reporting()。

语法

```
array_walk(array,function,userdata...)
```

参数	描述
array	必需。规定数组。
function	必需。用户自定义函数的名称。
userdata	可选。用户输入的值，可作为回调函数的参数。

提示和注释

提示：您可以为函数设置一个或多个参数。

注释：如果回调函数需要直接作用于数组中的值，可以将回调函数的第一个参数指定为引用：&$value。（参见例子 3）

注释：将键名和 userdata 传递到 function 中是 PHP 4.0 新增加的。

例子 1

```
<?php
function myfunction($value,$key)
{
echo "The key $key has the value $value<br />";
}
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
array_walk($a,"myfunction");
?>
```

输出：

```
The key a has the value Cat
The key b has the value Dog
The key c has the value Horse
```

例子 2

带有一个参数：

```
<?php
function myfunction($value,$key,$p)
{
echo "$key $p $value<br />";
}
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
array_walk($a,"myfunction","has the value");
?>
```

输出：

```
a has the value Cat
b has the value Dog
c has the value Horse
```

例子 3

改变数组元素的值（请注意 &$value）：

```
<?php
function myfunction(&$value,$key)
{
$value="Bird;
}
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
array_walk($a,"myfunction");
print_r($a);
?>
```

输出：

```
Array ( [a] => Bird [b] => Bird [c] => Bird )
```

PHP array_walk_recursive() 函数

定义和用法

与 array_walk() 函数 类似，array_walk_recursive() 函数对数组中的每个元素应用回调函数。不一样的是，如果原数组中的元素也是数组，就会递归地调用回调函数，也就是说，会递归到更深层的数组中去。

典型情况下，*function* 接受两个参数。*array* 参数的值作为第一个，键名作为第二个。如果提供了可选参数 *userdata* ，将被作为第三个参数传递给回调函数。

如果回调函数需要直接作用于数组中的值，可以将回调函数的第一个参数指定为引用，这样对这些单元的任何改变也将会改变原始数组本身。

语法

```
array_walk_recursive(array,function,userdata)
```

参数	描述
array	必需。规定数组。
function	必需。用户自定义函数的名称。
userdata	可选。用户输入的值，可作为回调函数的参数。

例子

```
<?php
function myfunction($value,$key)
{
echo "The key $key has the value $value<br />";
}
$a1=array("a"=>"Cat","b"=>"Dog");
$a2=array($a1,"1"=>"Bird","2"=>"Horse");
array_walk_recursive($a2,"myfunction");
?>
```

输出：

```
The key a has the value Cat
The key b has the value Dog
The key 1 has the value Bird
The key 2 has the value Horse
```

PHP arsort() 函数

定义和用法

arsort() 函数对数组进行逆向排序并保持索引关系。主要用于对那些单元顺序很重要的结合数组进行排序。

可选的第二个参数包含了附加的排序标识。

如果成功则返回 TRUE，否则返回 FALSE。

语法

```
arsort(array,sorttype)
```

参数	描述
array	必需。输入的数组。
sorttype	可选。规定如何排列数组的值。可能的值： SORT_REGULAR - 默认。以它们原来的类型进行处理（不改变类型）。 SORT_NUMERIC - 把值作为数字来处理 SORT_LOCALE_STRING - 把值作为字符串来处理，基于本地设置*。

*：该值是 PHP 4.4.0 和 5.0.2 新加的。在 PHP 6 之前，使用了系统的区域设置，可以用 setlocale() 来改变。自 PHP 6 起，必须用 i18n_loc_set_default() 函数。

例子

```
<?php
$my_array = array("a" => "Dog", "b" => "Cat", "c" => "Horse");

arsort($my_array);
print_r($my_array);
?>
```

输出：

```
Array
(
[c] => Horse
[a] => Dog
[b] => Cat
)
```

PHP asort() 函数

定义和用法

asort() 函数对数组进行排序并保持索引关系。主要用于对那些单元顺序很重要的结合数组进行排序。

可选的第二个参数包含了附加的排序标识。

如果成功则返回 TRUE，否则返回 FALSE。

语法

```
asort(array,sorttype)
```

参数	描述
array	必需。输入的数组。
sorttype	可选。规定如何排列数组的值。可能的值： `SORT_REGULAR` - 默认。以它们原来的类型进行处理（不改变类型）。 `SORT_NUMERIC` - 把值作为数字来处理 `SORT_STRING` - 把值作为字符串来处理 `SORT_LOCALE_STRING` - 把值作为字符串来处理，基于本地设置*。

*：该值是 PHP 4.4.0 和 5.0.2 新加的。在 PHP 6 之前，使用了系统的区域设置，可以用 setlocale() 来改变。自 PHP 6 起，必须用 i18n_loc_set_default() 函数。

例子

```
<?php
$my_array = array("a" => "Dog", "b" => "Cat", "c" => "Horse");

asort($my_array);
print_r($my_array);
?>
```

输出：

```
Array
(
[b] => Cat
[a] => Dog
[c] => Horse
)
```

PHP compact() 函数

定义和用法

compact() 函数创建一个由参数所带变量组成的数组。如果参数中存在数组，该数组中变量的值也会被获取。

本函数返回的数组是一个关联数组，键名为函数的参数，键值为参数中变量的值。

本函数执行的行为与 extract() 正好相反。

语法

```
compact(var1,var2...)
```

参数	描述
var1	必需。可以是带有变量名的字符串，或者是一个变量数组。
var2	可选。可以是带有变量名的字符串，或者是一个变量数组。允许多个参数。

注释和提示

注释：任何没有变量名与之对应的字符串都被略过。

例子 1

```
<?php
$firstname = "Peter";
$lastname = "Griffin";
$age = "38";

$result = compact("firstname", "lastname", "age");

print_r($result);
?>
```

输出：

```
Array
(
[firstname] => Peter
[lastname] => Griffin
[age] => 38
)
```

例子 2

使用没有对应变量名的字符串，以及一个变量名数组：

```
<?php
$firstname = "Peter";
$lastname = "Griffin";
$age = "38";

$name = array("firstname", "lastname");
$result = compact($name, "location", "age");

print_r($result);
?>
```

输出：

```
Array
(
[firstname] => Peter
[lastname] => Griffin
[age] => 38
)
```

PHP count() 函数

定义和用法

count() 函数计算数组中的单元数目或对象中的属性个数。

对于数组，返回其元素的个数，对于其他值，返回 1。如果参数是变量而变量没有定义，则返回 0。如果 mode 被设置为 COUNT_RECURSIVE（或 1），则会递归底计算多维数组中的数组的元素个数。

语法

```
count(array,mode)
```

参数	描述
array	必需。规定要计数的数组或对象。
mode	可选。规定函数的模式。可能的值：`0` - 默认。不检测多维数组（数组中的数组）。`1` - 检测多维数组。注释：该参数是 PHP 4.2 中加入的。

提示和注释

注释：当变量未被设置，或是变量包含一个空的数组，该函数会返回 0。可使用 isset() 变量来测试变量是否被设置。

例子

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
$result = count($people);

echo $result;
?>
```

输出：

```
4
```

PHP current() 函数

定义和用法

current() 函数返回数组中的当前元素（单元）。

每个数组中都有一个内部的指针指向它"当前的"元素，初始指向插入到数组中的第一个元素。

current() 函数返回当前被内部指针指向的数组元素的值，并不移动指针。如果内部指针指向超出了单元列表的末端，current() 返回 FALSE。

语法

```
current(array)
```

参数	描述
array	必需。规定要使用的数组。

提示和注释

注释：如果有空的元素，或元素没有值，该函数也返回 FALSE。

提示：该函数不会移动内部指针。要做到这一点，请使用 next() 和 prev() 函数。

例子

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

echo current($people) . "<br />";
?>
```

输出：

```
Peter
```

PHP each() 函数

定义和用法

each() 函数生成一个由数组当前内部指针所指向的元素的键名和键值组成的数组，并把内部指针向前移动。

返回的数组中包括的四个元素：键名为 0，1，key 和 value。单元 0 和 key 包含有数组单元的键名，1 和 value 包含有数据。

如果内部指针越过了数组范围，本函数将返回 FALSE。

语法

```
each(array)
```

参数	描述
array	必需。规定要使用的数组。

例子 1

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
print_r (each($people));
?>
```

输出：

```
Array ( [1] => Peter [value] => Peter [0] => 0 [key] => 0 )
```

例子 2

each() 经常和 list() 结合使用来遍历数组。本例与上例类似，不过循环输出了整个数组：

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

reset($people);

while (list($key, $val) = each($people))
  {
  echo "$key => $val<br />";
  }
?>
```

输出：

```
0 => Peter
1 => Joe
2 => Glenn
3 => Cleveland
```

例子解释

因为将一个数组赋值给另一个数组时会重置原来的数组指针，因此在上例中如果我们在循环内部将 $people 赋给了另一个变量的话将会导致无限循环。

PHP extract() 函数

定义和用法

PHP extract() 函数从数组中把变量导入到当前的符号表中。

对于数组中的每个元素，键名用于变量名，键值用于变量值。

第二个参数 type 用于指定当某个变量已经存在，而数组中又有同名元素时，extract() 函数如何对待这样的冲突。

本函数返回成功设置的变量数目。

语法

```
extract(array,extract_rules,prefix)
```

参数	描述
array	必需。规定要使用的输入。
extract_rules	可选。extract() 函数将检查每个键名是否为合法的变量名，同时也检查和符号表中的变量名是否冲突。对非法、数字和冲突的键名的处理将根据此参数决定。可以是以下值之一：可能的值：EXTR_OVERWRITE - 默认。如果有冲突，则覆盖已有的变量。 `EXTR_SKIP` - 如果有冲突，不覆盖已有的变量。（忽略数组中同名的元素） `EXTR_PREFIX_SAME` - 如果有冲突，在变量名前加上前缀 prefix。自 PHP 4.0.5 起，这也包括了对数字索引的处理。 `EXTR_PREFIX_ALL` - 给所有变量名加上前缀 prefix（第三个参数）。 `EXTR_PREFIX_INVALID` - 仅在非法或数字变量名前加上前缀 prefix。本标记是 PHP 4.0.5 新加的。 `EXTR_IF_EXISTS` - 仅在当前符号表中已有同名变量时，覆盖它们的值。其它的都不处理。可以用在已经定义了一组合法的变量，然后要从一个数组例如 $_REQUEST 中提取值覆盖这些变量的场合。本标记是 PHP 4.2.0 新加的。 `EXTR_PREFIX_IF_EXISTS` - 仅在当前符号表中已有同名变量时，建立附加了前缀的变量名，其它的都不处理。本标记是 PHP 4.2.0 新加的。 `EXTR_REFS` - 将变量作为引用提取。这有力地表明了导入的变量仍然引用了 var_array 参数的值。可以单独使用这个标志或者在 extract_type 中用 OR 与其它任何标志结合使用。本标记是 PHP 4.3.0 新加的。
prefix	可选。请注意 prefix 仅在 extract_type 的值是 `EXTR_PREFIX_SAME`， `EXTR_PREFIX_ALL`， `EXTR_PREFIX_INVALID` 或 `EXTR_PREFIX_IF_EXISTS` 时需要。如果附加了前缀后的结果不是合法的变量名，将不会导入到符号表中。前缀和数组键名之间会自动加上一个下划线。

例子 1

```
<?php
$a = 'Original';
$my_array = array("a" => "Cat","b" => "Dog", "c" => "Horse");
extract($my_array);
echo "\$a = $a; \$b = $b; \$c = $c";
?>
```

输出：

```
$a = Cat; $b = Dog; $c = Horse
```

例子 2

使用全部参数：

```
<?php
$a = 'Original';
$my_array = array("a" => "Cat","b" => "Dog", "c" => "Horse");

extract($my_array, EXTR_PREFIX_SAME, 'dup');

echo "\$a = $a; \$b = $b; \$c = $c; \$dup_a = $dup_a;";
?>
```

输出：

```
$a = Original; $b = Dog; $c = Horse; $dup_a = Cat;
```

PHP in_array() 函数

定义和用法

in_array() 函数在数组中搜索给定的值。

语法

```
in_array(value,array,type)
```

参数	描述
value	必需。规定要在数组搜索的值。
array	必需。规定要搜索的数组。
type	可选。如果设置该参数为 true，则检查搜索的数据与数组的值的类型是否相同。

说明

如果给定的值 *value* 存在于数组 *array* 中则返回 true。如果第三个参数设置为 true，函数只有在元素存在于数组中且数据类型与给定值相同时才返回 true。

如果没有在数组中找到参数，函数返回 false。

注释：如果 *value* 参数是字符串，且 *type* 参数设置为 true，则搜索区分大小写。

例子 1

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

if (in_array("Glenn",$people))
  {
  echo "Match found";
  }
else
  {
  echo "Match not found";
  }
?>
```

输出：

```
Match found
```

例子 2

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland", 23);

if (in_array("23",$people, TRUE))
  {
  echo "Match found<br />";
  }
else
  {
  echo "Match not found<br />";
  }if (in_array("Glenn",$people, TRUE))
  {
  echo "Match found<br />";
  }
else
  {
  echo "Match not found<br />";
  }if (in_array(23,$people, TRUE))
  {
  echo "Match found<br />";
  }
else
  {
  echo "Match not found<br />";
  }
?>
```

输出：

```
Match not found
Match found
Match found
```

PHP key() 函数

定义和用法

key() 函数返回数组内部指针当前指向元素的键名。

若失败，则返回 FALSE。

该函数与 current() 类似，只是返回的结果不同。current() 函数返回的是元素的值，而 key() 函数返回的是元素的键名。

语法

```
key(array)
```

参数	描述
array	必需。规定要使用的数组。

例子

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo "The key from the current position is: " . key($people);
?>
```

输出：

```
The key from the current position is: 0
```

PHP krsort() 函数

定义和用法

krsort() 函数将数组按照键逆向排序，为数组值保留原来的键。

可选的第二个参数包含附加的排序标志。

若成功，则返回 TRUE，否则返回 FALSE。

语法

```
krsort(array,sorttype)
```

参数	描述
array	必需。规定要排序的数组。
sorttype	可选。规定如何排列数组的值。可能的值： SORT_REGULAR - 默认。以它们原来的类型进行处理（不改变类型）。 SORT_NUMERIC - 把值作为数字来处理 SORT_STRING - 把值作为字符串来处理 SORT_LOCALE_STRING - 把值作为字符串来处理，基于本地设置*。

*：该值是 PHP 4.4.0 和 5.0.2 新加的。在 PHP 6 之前，使用了系统的区域设置，可以用 setlocale() 来改变。自 PHP 6 起，必须用 i18n_loc_set_default() 函数。

例子

```
<?php
$my_array = array("a" => "Dog", "b" => "Cat", "c" => "Horse");
krsort($my_array);
print_r($my_array);
?>
```

输出：

```
Array
(
[c] => Horse
[b] => Cat
[a] => Dog
)
```

PHP ksort() 函数

定义和用法

ksort() 函数按照键名对数组排序，为数组值保留原来的键。

可选的第二个参数包含附加的排序标志。

若成功，则返回 TRUE，否则返回 FALSE。

语法

```
ksort(array,sorttype)
```

参数	描述
array	必需。规定要排序的数组。
sorttype	可选。规定如何排列数组的值。可能的值： SORT_REGULAR - 默认。以它们原来的类型进行处理（不改变类型）。 SORT_NUMERIC - 把值作为数字来处理 SORT_STRING - 把值作为字符串来处理 SORT_LOCALE_STRING - 把值作为字符串来处理，基于本地设置*。

*：该值是 PHP 4.4.0 和 5.0.2 新加的。在 PHP 6 之前，使用了系统的区域设置，可以用 setlocale() 来改变。自 PHP 6 起，必须用 i18n_loc_set_default() 函数。

例子

```
<?php
$my_array = array("a" => "Dog", "b" => "Cat", "c" => "Horse");
ksort($my_array);
print_r($my_array);
?>
```

输出：

```
Array
(
[a] => Dog
[b] => Cat
[c] => Horse
)
```

PHP list() 函数

定义和用法

list() 函数用数组中的元素为一组变量赋值。

注意，与 array() 类似，list() 实际上是一种语言结构，不是函数。

语法

```
list(var1,var2...)
```

参数	描述
var1	必需。第一个需要赋值的变量。
var2	可选。可以有多个变量。

提示和注释

注释：该函数只用于数字索引的数组，且假定数字索引从 0 开始。

例子 1

```
<?php
$my_array = array("Dog","Cat","Horse");

list($a, $b, $c) = $my_array;
echo "I have several animals, a $a, a $b and a $c.";
?>
```

输出：

```
I have several animals, a Dog, a Cat and a Horse.
```

例子 2

```
<?php
$my_array = array("Dog","Cat","Horse");

list($a, , $c) = $my_array;
echo "Here I only use the $a and $c variables.";
?>
```

输出：

```
Here I only use the Dog and Horse variables.
```

PHP natcasesort() 函数

定义和用法

natcasesort() 函数用不区分大小写的自然顺序算法对给定数组中的元素排序。

natcasesort() 函数实现了“自然排序”，即数字从 1 到 9 的排序方法，字母从 a 到 z 的排序方法，短者优先，该函数不区分大小写。数组的索引与单元值保持关联。

如果成功，则该函数返回 TRUE，否则返回 FALSE。

语法

```
natcasesort(array)
```

参数	描述
array	必需。规定要进行排序的数组。

提示和注释

提示：natcasesort() 是 natsort() 函数 的不区分大小写字母的版本。

例子

```
<?php
$temp_files = array("temp15.txt","Temp10.txt",
"temp1.txt","Temp22.txt","temp2.txt");

natsort($temp_files);
echo "Natural order: ";
print_r($temp_files);
echo "<br />";

natcasesort($temp_files);
echo "Natural order case insensitve: ";
print_r($temp_files);
?>
```

输出：

```
Natural order:
Array
(
[0] => Temp10.txt
[1] => Temp22.txt
[2] => temp1.txt
[4] => temp2.txt
[3] => temp15.txt
)

Natural order case insensitve:
Array
(
[2] => temp1.txt
[4] => temp2.txt
[0] => Temp10.txt
[3] => temp15.txt
[1] => Temp22.txt
)
```

PHP natsort() 函数

定义和用法

natsort() 函数用自然顺序算法对给定数组中的元素排序。

natsort() 函数实现了“自然排序”，即数字从 1 到 9 的排序方法，字母从 a 到 z 的排序方法，短者优先。数组的索引与单元值保持关联。

如果成功，则该函数返回 TRUE，否则返回 FALSE。

语法

```
natsort(array)
```

参数	描述
array	必需。规定要进行排序的数组。

例子

本函数所用的自然排序算法，与通常的计算机字符串排序算法（用于 sort()）的区别，见下面示例：

```
<?php
$temp_files = array("temp15.txt","temp10.txt",
"temp1.txt","temp22.txt","temp2.txt");

sort($temp_files);
echo "Standard sorting: ";
print_r($temp_files);
echo "<br />";

natsort($temp_files);
echo "Natural order: ";
print_r($temp_files);
?>
```

输出：

```
Standard sorting: Array
(
[0] => temp1.txt
[1] => temp10.txt
[2] => temp15.txt
[3] => temp2.txt
[4] => temp22.txt
)

Natural order: Array
(
[0] => temp1.txt
[3] => temp2.txt
[1] => temp10.txt
[2] => temp15.txt
[4] => temp22.txt
)
```

PHP next() 函数

定义和用法

next() 函数把指向当前元素的指针移动到下一个元素的位置，并返回当前元素的值。

如果内部指针已经超过数组的最后一个元素，函数返回 false。

语法

```
next(array)
```

参数	描述
array	必需。规定要使用的数组。

说明

next() 和 current() 的行为类似，只有一点区别，在返回值之前将内部指针向前移动一位。这意味着它返回的是下一个数组单元的值并将数组指针向前移动了一位。如果移动指针的结果超出了数组单元的末端，则 next() 返回 FALSE。

注意：如果数组包含空的单元，或者单元的值是 0 则该函数碰到这些单元也返回 FALSE。要正确遍历可能含有空单元或者单元值为 0 的数组，请参见 each() 函数。

例子

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

echo current($people) . "<br />";
echo next($people);
?>
```

输出：

```
Peter
Joe
```

PHP pos() 函数

定义和用法

pos() 函数是 current() 函数 的别名。它可返回数组中当前元素的值。

语法

```
pos(array)
```

参数	描述
array	必需。规定要使用的数组。

例子

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo pos($people) . "<br />";
?>
```

输出：

```
Peter
```

PHP prev() 函数

定义和用法

prev() 函数把指向当前元素的指针移动到上一个元素的位置，并返回当前元素的值。

如果内部指针已经超过数组的第一个元素之前，函数返回 false。

语法

```
prev(_array_)
```

参数	描述
array	必需。规定要使用的数组。

说明

prev() 和 next() 的行为类似，不过它将内部指针倒回一位而不是前移一位。

注意：如果数组包含空的单元，或者单元的值是 0 则该函数碰到这些单元也返回 FALSE。要正确遍历可能含有空单元或者单元值为 0 的数组，请参见 each() 函数。

例子

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

echo current($people) . "<br />";
echo next($people) . "<br />";
echo prev($people);
?>
```

输出：

```
Peter
Joe
Peter
```

PHP range() 函数

定义和用法

range() 函数创建并返回一个包含指定范围的元素的数组。

语法

```
range(first,second,step)
```

参数	描述
first	必需。规定数组元素的最小值。
second	必需。规定数组元素的最大值。
step	可选。规定元素之间的步进制。默认是 1。注释：该参数是 PHP 5 中加入的。

说明

该函数创建一个数组，包含从 first 到 second （包含 first 和 second）之间的整数或字符。如果 second 比 first 小，则返回反序的数组。

例子 1

```
<?php
$number = range(0,5);
print_r ($number);
?>
```

输出：

```
Array
(
[0] => 0
[1] => 1
[2] => 2
[3] => 3
[4] => 4
[5] => 5
)
```

例子 2

```
<?php
$number = range(0,50,10);
print_r ($number);
?>
```

输出：

```
Array
(
[0] => 0
[1] => 10
[2] => 20
[3] => 30
[4] => 40
[5] => 50
)
```

例子 3

```
<?php
$letter = range("a","d");
print_r ($letter);
?>
```

输出：

```
Array
(
[0] => a
[1] => b
[2] => c
[3] => d
)
```

PHP reset() 函数

定义和用法

reset() 函数把数组的内部指针指向第一个元素，并返回这个元素的值。

若失败，则返回 FALSE。

语法

```
reset(array)
```

参数	描述
array	必需。规定要使用的数组。

例子

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

echo current($people) . "<br />";
echo next($people) . "<br />";
echo reset($people);
?>
```

输出：

```
Peter
Joe
Peter
```

PHP rsort() 函数

定义和用法

rsort() 函数对数组的元素按照键值进行逆向排序。与 arsort() 的功能基本相同。

注释：该函数为 array 中的单元赋予新的键名。这将删除原有的键名而不仅是重新排序。

如果成功则返回 TRUE，否则返回 FALSE。

可选的第二个参数包含另外的排序标志。

语法

```
rsort(array,sorttype)
```

参数	描述
array	必需。输入的数组。
sorttype	可选。规定如何排列数组的值。可能的值：`SORT_REGULAR` - 默认。以它们原来的类型进行处理（不改变类型）。`SORT_NUMERIC` - 把值作为数字来处理 `SORT_STRING` - 把值作为字符串来处理 `SORT_LOCALE_STRING` - 把值作为字符串来处理，基于本地设置*。

*：该值是 PHP 4.4.0 和 5.0.2 新加的。在 PHP 6 之前，使用了系统的区域设置，可以用 setlocale() 来改变。自 PHP 6 起，必须用 i18n_loc_set_default() 函数。

例子

```
<?php
$my_array = array("a" => "Dog", "b" => "Cat", "c" => "Horse");
rsort($my_array);
print_r($my_array);
?>
```

输出：

```
Array
(
[0] => Horse
[1] => Dog
[2] => Cat
)
```

PHP shuffle() 函数

定义和用法

shuffle() 函数把数组中的元素按随机顺序重新排列。

若成功，则返回 TRUE，否则返回 FALSE。

注释：本函数为数组中的单元赋予新的键名。这将删除原有的键名而不仅是重新排序。

注释：自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 函数给随机数发生器播种，现已被自动完成。

语法

```
shuffle(array)
```

参数	描述
array	必需。规定要使用的数组。

例子

```
<?php
$my_array = array("a" => "Dog", "b" => "Cat", "c" => "Horse");
shuffle($my_array);
print_r($my_array);
?>
```

输出：

```
Array ( [0] => Cat [1] => Horse [2] => Dog )
```

PHP sizeof() 函数

定义和用法

sizeof() 函数计算数组中的单元数目或对象中的属性个数。

该函数是 count() 的别名。

语法

```
sizeof(array,mode)
```

参数	描述
array	必需。规定要计数的数组或对象。
mode	可选。规定函数的模式。可能的值： `0` - 默认。不检测多维数组（数组中的数组）。 `1` - 检测多维数组。注释：该参数是 PHP 4.2 中加入的。

提示和注释

注释：当变量未被设置，或是变量包含一个空的数组，该函数会返回 0。可使用 isset() 变量来测试变量是否被设置。

例子

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
$result = sizeof($people);

echo $result;
?>
```

输出：

```
4
```

PHP sort() 函数

定义和用法

sort() 函数按升序对给定数组的值排序。

注释：本函数为数组中的单元赋予新的键名。原有的键名将被删除。

如果成功则返回 TRUE，否则返回 FALSE。

语法

```
sort(_array_,_sorttype_)
```

参数	描述
array	必需。输入的数组。
sorttype	可选。规定如何排列数组的值。可能的值： `SORT_REGULAR` - 默认。以它们原来的类型进行处理（不改变类型）。 `SORT_NUMERIC` - 把值作为数字来处理 `SORT_STRING` - 把值作为字符串来处理 `SORT_LOCALE_STRING` - 把值作为字符串来处理，基于本地设置*。

*：该值是 PHP 4.4.0 和 5.0.2 新加的。在 PHP 6 之前，使用了系统的区域设置，可以用 setlocale() 来改变。自 PHP 6 起，必须用 i18n_loc_set_default() 函数。

例子

```
<?php
$my_array = array("a" => "Dog", "b" => "Cat", "c" => "Horse");

sort($my_array);
print_r($my_array);
?>
```

输出：

```
Array
(
[0] => Cat
[1] => Dog
[2] => Horse
)
```

PHP uasort() 函数

定义和用法

uasort() 函数使用用户自定义的比较函数对数组排序，并保持索引关联（不为元素分配新的键）。

如果成功则返回 TRUE，否则返回 FALSE。

该函数主要用于对那些单元顺序很重要的结合数组进行排序。

语法

```
uasort(array,sorttype)
```

参数	描述
array	必需。规定要排序的数组。
function	

必需。用户自定义的函数。

函数必须设计为返回 -1, 0, 或 1，并应该接受两个供比较的参数，同时以类似下面这样的方式来工作： 如果 a = b, 返回 0； 如果 a < b, 返回 1； 如果 a > b, 返回 -1。 |

例子

```
<?php
function my_sort($a, $b)
  {
  if ($a == $b) return 0;
  return ($a > $b) ? -1 : 1;
  }

$people = array("Swanson" => "Joe",
"Griffin" => "Peter", "Quagmire" => "Glenn",
"swanson" => "joe", "griffin" => "peter",
"quagmire" => "glenn");

uasort($people, "my_sort");

print_r ($people);
?>
```

输出：

```
Array
(
[griffin] => peter
[swanson] => joe
[quagmire] => glenn
[Griffin] => Peter
[Swanson] => Joe
[Quagmire] => Glenn
)
```

PHP uksort() 函数

定义和用法

uksort() 函数使用用户自定义的比较函数按照键名对数组排序，并保持索引关系。

如果成功则返回 TRUE，否则返回 FALSE。

如果要排序的数组需要用一种不寻常的标准进行排序，那么应该使用此函数。

自定义函数应接受两个参数，该参数将被数组中的一对键名填充。比较函数在第一个参数小于，等于，或大于第二个参数时必须分别返回一个小于零，等于零，或大于零的整数。

语法

```
uksort(array,sorttype)
```

参数	描述
array	必需。规定要排序的数组。
function	必需。用户自定义的函数。函数必须设计为返回 -1, 0, 或 1，并应该接受两个供比较的参数，同时以类似下面这样的方式来工作： 如果 a = b, 返回 0； 如果 a > b, 返回 1； 如果 a < b, 返回 -1。

例子

```
<?php
function my_sort($a, $b)
  {
  if ($a == $b) return 0;
  return ($a > $b) ? -1 : 1;
  }

$people = array("Swanson" => "Joe",
"Griffin" => "Peter", "Quagmire" => "Glenn",
"swanson" => "joe", "griffin" => "peter",
"quagmire" => "glenn");

uksort($people, "my_sort");

print_r ($people);
?>
```

输出：

```
Array
(
[swanson] => joe
[quagmire] => glenn
[griffin] => peter
[Swanson] => Joe
[Quagmire] => Glenn
[Griffin] => Peter
)
```

PHP usort() 函数

定义和用法

usort() 函数使用用户自定义的函数对数组排序。

注释：如果两个元素比较结果相同，则它们在排序后的数组中的顺序未经定义。到 PHP 4.0.6 之前，用户自定义函数将保留这些元素的原有顺序。但是由于在 4.1.0 中引进了新的排序算法，结果将不是这样了，因为对此没有一个有效的解决方案。

注释：本函数为 *array* 中的元素赋予新的键名。这会删除原有的键名。

语法

```
usort(array,sorttype)
```

参数	描述
array	必需。规定要排序的数组。
function	必需。用户自定义的函数。函数必须设计为返回 -1, 0, 或 1，并应该接受两个供比较的参数，同时以类似下面这样的方式来工作： 如果 a = b, 返回 0； 如果 a > b, 返回 1； 如果 a < b, 返回 -1。

例子

```
<?php
function my_sort($a, $b)
  {
  if ($a == $b) return 0;
  return ($a > $b) ? -1 : 1;
  }

$arr = array("Peter", "glenn","Cleveland","peter","cleveland", "Glenn");

usort($arr, "my_sort");

print_r ($arr);
?>
```

输出：

```
Array
(
[0] => peter
[1] => glenn
[2] => cleveland
[3] => Peter
[4] => Glenn
[5] => Cleveland
)
```

PHP Calendar 函数

PHP Calendar 简介

当使用不同的历法格式时，calendar 函数很有用。它所基于的标准是儒略日计数 (Julian day count)。

编者注：Julian day count 是从 January 1, 4713 B.C. 开始计算的，中文译为儒略日计数或恺撒日计数。

请注意，Julian day count（儒略日计数） 与 Julian calendar（儒略历） 不是一回事。

注释：如需在日历格式之间转换，必须首先转换为 Julian day count，然后再转换为日历格式。

安装

PHP 的 windows 版本已内建了对日历扩展的支持。因此，Calendar 函数会自动工作。

不过，如果您运行的是 PHP 的 Linux 版本，就不得不通过 --enable-calendar 编译 PHP，这样日历函数才能工作。

PHP Calendar 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
cal_days_in_month()	针对指定的年份和日历，返回一个月中的天数。	4
cal_from_jd()	把儒略日计数转换为指定日历的日期。	4
cal_info()	返回有关给定日历的信息。	4
cal_to_jd()	把日期转换为儒略日计数。	4
easter_date()	返回指定年份的复活节午夜的 Unix 时间戳。	3
easter_days()	返回指定年份的复活节与 3 月 21 日之间的天数。	3
FrenchToJD()	将法国共和历法转换成为儒略日计数。	3
GregorianToJD()	将格利高里历法转换成为儒略日计数。	3
JDDayOfWeek()	返回日期在周几。	3
JDMonthName()	返回月的名称。	3
JDToFrench()	把儒略日计数转换为法国共和国历法。	3
JDToGregorian()	把儒略日计数转换为格利高里历法。	3
jdtojewish()	把儒略日计数转换为犹太历法。	3
JDToJulian()	把儒略日计数转换为儒略历。	3
jdtounix()	把儒略日计数转换为 Unix 时间戳。	4
JewishToJD()	把犹太历法转换为儒略日计数。	3
JulianToJD()	把儒略历转换为儒略日计数。	3
unixtojd()	把 Unix 时间戳转换为儒略日计数。	4

PHP Calendar 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
CAL_GREGORIAN	Gregorian calendar	3
CAL_JULIAN	Julian calendar	3
CAL_JEWISH	Jewish calendar	3
CAL_FRENCH	French Republican calendar	3
CAL_NUM_CALS	3	
CAL_DOW_DAYNO	3	
CAL_DOW_SHORT	3	
CAL_DOW_LONG	3	
CAL_MONTH_GREGORIAN_SHORT	3	
CAL_MONTH_GREGORIAN_LONG	3	
CAL_MONTH_JULIAN_SHORT	3	
CAL_MONTH_JULIAN_LONG	3	
CAL_MONTH_JEWISH	3	
CAL_MONTH_FRENCH	3	
CAL_EASTER_DEFAULT	4	
CAL_EASTER_DEFAULT	4	
CAL_EASTER_ROMAN	4	
CAL_EASTER_ALWAYS_GREGORIAN	4	
CAL_EASTER_ALWAYS_JULIAN	4	
CAL_JEWISH_ADD_ALAFIM_GERESH	5	
CAL_JEWISH_ADD_ALAFIM	5	
CAL_JEWISH_ADD_GERESHAYIM	5	

PHP cal_days_in_month() 函数

定义和用法

cal_days_in_month() 函数针对指定的年份和日历，返回一个月中的天数。

语法

```
cal_days_in_month(calendar,month,year)
```

参数	描述
calendar	必需。规定要使用的历法。
month	必须。规定月。
year	必须。规定年。

例子

```
<?php
$d=cal_days_in_month(CAL_GREGORIAN,10,2005);
echo("There was $d days in October 2005");
?>
```

输出：

```
There was 31 days in October 2005
```

PHP cal_from_jd() 函数

定义和用法

cal_from_jd() 函数把儒略日计数转换为指定历法的日期。

语法

```
cal_from_jd(jd,calendar)
```

参数	描述
jd	必需。一个数字（儒略日计数）。
calendar	必需。规定要使用的历法。可以使用下面这些常量： CAL_GREGORIAN CAL_JULIAN CAL_JEWISH CAL_FRENCH

例子

```
<?php
$d=unixtojd(mktime(0,0,0,1,18,2006));
print_r(cal_from_jd($d,CAL_GREGORIAN));
?>
```

输出：

```
Array
(
[date] => 1/18/2006
[month] => 1
[day] => 18
[year] => 2006
[dow] => 3
[abbrevdayname] => Wed
[dayname] => Wednesday
[abbrevmonth] => Jan
[monthname] => January
)
```

PHP cal_info() 函数

定义和用法

cal_info() 函数返回一个数组，其中包含了关于给定历法的信息。

所返回的数组包含这些元素：calname, calsymbol, month, abbrevmonth 以及 maxdaysinmonth。

语法

```
cal_from_jd(jd,calendar)
```

参数	描述
calendar	必需。规定要使用的历法。可以使用下面这些常量：`CAL_GREGORIAN` `CAL_JULIAN` `CAL_JEWISH` `CAL_FRENCH`

提示和注释

提示：在 PHP 5 中，如果没有指定 calendar 参数，则返回所有被支持的历法的信息。

例子

```
<?php
$calinfo=cal_info(0);
print_r($calinfo);
?>
```

输出：

```
Array
(
[months] => Array
(
[1] => January
[2] => February
[3] => March
[4] => April
[5] => May
[6] => June
[7] => July
[8] => August
[9] => September
[10] => October
[11] => November
[12] => December
)

[abbrevmonths] => Array
(
[1] => Jan
[2] => Feb
[3] => Mar
[4] => Apr
[5] => May
[6] => Jun
[7] => Jul
[8] => Aug
[9] => Sep
[10] => Oct
[11] => Nov
[12] => Dec
)

[maxdaysinmonth] => 31
[calname] => Gregorian
[calsymbol] => CAL_GREGORIAN
)
```

PHP cal_to_jd() 函数

定义和用法

cal_to_jd() 函数把指定的日期转换为儒略日计数。

语法

```
cal_to_jd(calendar,month,day,year)
```

参数	描述
calendar	必需。规定要使用的历法。可以使用下面这些常量：`CAL_GREGORIAN` `CAL_JULIAN` `CAL_JEWISH` `CAL_FRENCH`
month	必需。规定月。
day	必需。规定日。
year	必需。规定年。

例子

```
<?php
$d=cal_to_jd(CAL_GREGORIAN,10,03,2005);
echo($d);
?>
```

输出：

```
2453647
```

PHP easter_date() 函数

定义和用法

easter_date() 函数返回指定年份的复活节午夜的 Unix 时间戳。

输入一个年份，则以 UNIX 时间戳记的格式返回该年的复活节日期，若没有输入年份，则计算当年的日期。

语法

```
easter_date(year)
```

参数	描述
year	可选。定义用于计算复活节日期的年份。若省略，使用当年。

提示和注释

注释：如果年份在 Unix 时间戳的范围之外（1970 之前或 2037 之后），该函数会生成一个警告。可使用 easter_days() 代替 easter_date() 来计算年份在范围之外的复活节日期。

例子

```
<?php
echo(easter_date() . "<br />");
echo(date("M-d-Y",easter_date()) . "<br />");
echo(date("M-d-Y",easter_date(2000)) . "<br />");
echo(date("M-d-Y",easter_date(2001)) . "<br />");
echo(date("M-d-Y",easter_date(2002)));
?>
```

输出：

```
1145138400
Apr-16-2006
Apr-23-2000
Apr-15-2001
Mar-31-2002
```

PHP easter_days() 函数

定义和用法

easter_days() 函数返回指定年份的复活节与 3 月 21 日之间的天数。

输入一个年份，则计算该年复活节与三月二十一日之间的日期数，若没有输入年份，则以当年计算。这个函数可以用来替代 easter_date() 在 1970-2037 年范围外无法计算的问题。

语法

```
easter_date(year)
```

参数	描述
year	可选。定义用于计算复活节日期的年份。若省略，使用当年。
method	可选。允许你计算机与其它历法的复活节日期。例如，如果设置为 CAL_EASTER_ROMAN，则使用 1582 - 1752 年期间的格利高里历法。

例子

```
<?php
echo(easter_days() . "<br />");
echo(easter_days(1990) . "<br />");
echo(easter_days(1342) . "<br />");
echo(easter_days(2050);
?>
```

输出：

```
26
25
10
20
```

PHP FrenchToJD() 函数

定义和用法

FrenchToJD() 函数将法国共和历法转换成为儒略日计数。

语法

```
frenchtojd(month,day,year)
```

参数	描述
month	必需。规定月
day	必需。规定日
year	可选。必须在 1 到 14 的范围内。

提示和注释

法国共和历法是法国革命期间提出的一种历法，从 1793 年晚期开始，法国政府使用了大约 12 年。该函数只转换 1 到 14 年内的日期 （格利高里历 1792 年 9 月 22 日 - 1806 年 9 月 22 日）。

例子

```
<?php
$d=frenchtojd(3,3,14);
echo($d);
?>
```

输出：

```
2380650
```

PHP GregorianToJD() 函数

定义和用法

GregorianToJD() 函数将格利高里历法转换成为儒略日计数。

语法

```
gregoriantojd(month,day,year)
```

参数	描述
month	必需。规定月
day	必需。规定日
year	可选。合法的范围是 4714 B.C. 到 9999 A.D。

提示和注释

尽管该函数可处理 4714 B.C. 之前的日期，您还是要注意格利高里历法在 1582 年才建立，一些国家甚至更晚才接受它（大不列颠在 1752 年，苏联在 1918 年，希腊在 1923 年）。大部分欧洲国家使用罗马儒略历（公历）先于格利高里历法。

例子

```
<?php
$jd = gregoriantojd(10,3,1975);
echo($jd . "<br />");

$gregorian = jdtogregorian($jd);
echo($gregorian);
?>
```

输出：

```
2442689
10/3/1975
```

PHP JDDayOfWeek() 函数

定义和用法

JDDayOfWeek() 函数返回日期在周几。

语法

```
jddayofweek(jd,mode)
```

参数	描述
jd	必需。数字（儒略日计数）。
mode	可选。定义返回的内容（数字还是字符串）。模式值： 0 - 默认。以整数返回周的天。（0 为周日, 1 为周一... 余类推） 1 - 返回包含周的天的字符串。（英文-格里高里历） 2 - 返回包含周的天的简写的字符串。（英文-格里高里历）

例子

```
<?php
$jd=cal_to_jd(CAL_GREGORIAN,date("m"),date("d"),date("Y"));
echo(jddayofweek($jd,1));
?>
```

输出：

```
Thursday
```

PHP JDMonthName() 函数

定义和用法

JDMonthName() 函数返回指定历法的月份字符串。

语法

```
jdmonthname(jd,mode)
```

参数	描述
jd	必需。数字（儒略日计数）。
mode	可选。定义把儒略日计数转换为哪种历法，以及返回哪种月份名称。模式值： `0` - 格里高里历 (缩写) (Jan, Feb, Mar, ...) `1` - 格里高里历 (January, February, March, ...) `2` - 凯撒历 (缩写) (Jan, Feb, Mar, ...) `3` - 凯撒历 (January, February, March, ...) `4` - 犹太历 (Tishri, Heshvan, Kislev, ...) `5` - 法国共和历 (Vendemiaire, Brumaire, Frimaire, ...)

例子

```
<?php
$jd=cal_to_jd(CAL_GREGORIAN,date("m"),date("d"),date("Y"));
echo(jdmonthname($jd,1));
?>
```

输出：

```
January
```

PHP JDToFrench() 函数

定义和用法

JDToFrench() 函数把儒略日计数转换为法国共和国历法。

语法

```
jdtofrench(jd)
```

参数	描述
jd	必需。数字（儒略日计数）。

提示和注释：

注释：该函数以 "month/day/year" 的格式返回一个字符串。

提示：法国共和历法是法国革命期间提出的一种历法，从 1793 年晚期开始，法国政府使用了大约 12 年。

例子

```
<?php
$d=jdtofrench(2380650);
echo($d);
?>
```

输出：

```
3/3/14
```

PHP JDToGregorian() 函数

定义和用法

JDToGregorian() 函数把儒略日计数转换为格利高里历法。

语法

```
jdtogregorian(jd)
```

参数	描述
jd	必需。数字（儒略日计数）。

提示和注释：

注释：该函数以 "month/day/year" 的格式返回一个字符串。

例子

```
<?php
$jd = gregoriantojd(10,3,1975);
echo($jd . "<br />");

$gregorian = jdtogregorian($jd);
echo($gregorian);
?>
```

输出：

```
2442689
10/3/1975
```

PHP JDToJewish() 函数

定义和用法

JDToJewish() 函数把儒略日计数转换为犹太历法

语法

```
JDToJewish(jd)
```

参数	描述
jd	必需。数字（儒略日计数）。
hebrew	可选。True 指示希伯来语输出格式。
fl	可选。定义希伯来语输出格式，可用的格式有：`CAL_JEWISH_ADD_ALAFIM_GERESH` `CAL_JEWISH_ADD_ALAFIM` `CAL_JEWISH_ADD_GERESHAYIM`

例子

```
<?php
echo(jdtojewish(gregoriantojd(10,8,2002)));
?>
```

输出：

```
2/2/5763
```

PHP JDToJulian() 函数

定义和用法

JDToJulian() 函数把儒略日计数转换为儒略历。

语法

```
JDToJulian(jd)
```

参数	描述
jd	必需。数字（儒略日计数）。

提示和注释：

注释：该函数以 "month/day/year" 的格式返回一个字符串。

例子

```
<?php
$jd = juliantojd(10,3,1975);
echo($jd . "<br />");

$julian = jdtojulian($jd);
echo($julian);
?>
```

输出：

```
2442702
10/3/1975
```

PHP JDToUnix() 函数

定义和用法

JDToUnix() 函数把儒略日计数转换为 Unix 时间戳。

语法

```
JDToUnix(jd)
```

参数	描述
jd	必需。数字（儒略日计数）。

提示和注释：

注释：如果参数 jd 不在 Unix 新纪元之中（意味着格利高里年必须介于 1970 和 2037 之间，或者 jd >= 2440588 且 jd <= 2465342），则该函数将返回 false。所返回的时间是本地时间。

例子

```
<?php
$jd = gregoriantojd(10,3,1970);
$unix = jdtounix($jd);
echo($unix);
?>
```

输出：

```
23760000
```

PHP JewishToJD() 函数

定义和用法

JewishToJD() 函数把犹太历法转换为儒略日计数。

语法

```
JDToUnix(jd)
```

参数	描述
month	必需。规定月。
day	必需。规定日。
year	必需。规定年。

提示和注释：

注释：有效的范围为犹太历法公元前 3761 年起。犹太历法存在了数千年，但早期并没有公式化的开始月份计算法。每年的第一个月为首次观测到的新月。

例子

```
<?php
echo(jewishtojd(2,2,5763));
?>
```

输出：

```
2452556
```

PHP JulianToJD() 函数

定义和用法

JulianToJD() 函数把儒略历转换为儒略日计数。

语法

```
JDToUnix(jd)
```

参数	描述
month	必需。规定月。
day	必需。规定日。
year	必需。规定年。合法的范围是 4713 B.C. 到 9999 A.D。

提示和注释：

注释：有效的范围为凯撒历法公元前 4713 年至公元 9999 年。该函数能计算到公元前 4713 年，但这是不太有意义的。凯撒历法是在公元前 46 年建立的，但一些细节等到公元 8 年才稳定下来。

例子

```
<?php
$jd = juliantojd(10,3,1975);
echo($jd . "<br />");

$julian = jdtojulian($jd);
echo($julian);
?>
```

输出：

```
2442702
10/3/1975
```

PHP UnixToJD() 函数

定义和用法

UnixToJD() 函数把 Unix 时间戳转换为儒略日计数。

语法

```
unixtojd(timestamp)
```

参数	描述
timestamp	可选。

提示和注释：

注释：Unix 时间戳指示的是从格利高里历（不是罗马儒略历）的 1970 年 1 月 1 日至今的秒数。

例子

```
<?php
echo(unixtojd());
?>
```

输出：

```
2453755
```

PHP cURL 函数

概述

PHP支持的由Daniel Stenberg创建的libcurl库允许你与各种的服务器使用各种类型的协议进行连接和通讯。

libcurl目前支持http、https、ftp、gopher、telnet、dict、file和ldap协议。libcurl同时也支持HTTPS认证、HTTP POST、HTTP PUT、 FTP 上传(这个也能通过PHP的FTP扩展完成)、HTTP 基于表单的上传、代理、cookies和用户名+密码的认证。

PHP中使用cURL实现Get和Post请求的方法

这些函数在PHP 4.0.2中被引入。

需求

为了使用PHP的cURL函数，你需要安装 ? libcurl包。

PHP需要使用libcurl 7.0.2-beta 或者更高版本。在PHP 4.2.3 里使用cURL，你需要安装7.9.0或更高版本的libcurl。从PHP 4.3.0开始你需要安装7.9.0或更高版本的libcurl。从PHP 5.0.0开始你需要安装7.10.5或更高版本的libcurl。

安装

要使用PHP的cURL支持你必须在编译PHP时加上--with-curl[=DIR] 选项，DIR为包含lib和include的目录路径。在include目录中必须有一个名为curl，包含了easy.h和curl.h的文件夹。lib文件夹里应该有一个名为libcurl.a的文件。对于PHP 4.3.0你可以配置--with-curlwrappers 使cURL使用URL流。

注意: Win32用户注意 要在Windows环境下使用这个模块，libeay32.dll和ssleay32.dll必须放到PATH环境变量包含的目录下。 不用cURL网站上的libcurl.dll。

资源类型

这个扩展定义了2中资源：cURL句柄和cURL批处理句柄。

PHP cURL 函数

以下包含了PHP cURL函数列表：

函数	描述
curl_close()	关闭一个cURL会话。
curl_copy_handle()	复制一个cURL句柄和它的所有选项。
curl_errno()	返回最后一次的错误号。
curl_error()	返回一个保护当前会话最近一次错误的字符串。
curl_escape()	返回转义字符串，对给定的字符串进行URL编码。
curl_exec()	执行一个cURL会话。
curl_file_create()	创建一个 CURLFile 对象。
curl_getinfo()	获取一个cURL连接资源句柄的信息。
curl_init()	初始化一个cURL会话。
curl_multi_add_handle()	向curl批处理会话中添加单独的curl句柄。
curl_multi_close()	关闭一组cURL句柄。
curl_multi_exec()	运行当前 cURL 句柄的子连接。
curl_multi_getcontent()	如果设置了CURLOPT_RETURNTRANSFER，则返回获取的输出的文本流。
curl_multi_info_read()	获取当前解析的cURL的相关传输信息。
curl_multi_init()	返回一个新cURL批处理句柄。
curl_multi_remove_handle()	移除curl批处理句柄资源中的某个句柄资源。
curl_multi_select()	等待所有cURL批处理中的活动连接。
curl_multi_setopt()	设置一个批处理cURL传输选项。
curl_multi_strerror()	返回描述错误码的字符串文本。
curl_pause()	暂停及恢复连接。
curl_reset()	重置libcurl的会话句柄的所有选项。
curl_setopt_array()	为cURL传输会话批量设置选项。
curl_setopt()	设置一个cURL传输选项。
curl_share_close()	关闭cURL共享句柄。
curl_share_init()	初始化cURL共享句柄。
curl_share_setopt()	设置一个共享句柄的cURL传输选项。
curl_strerror()	返回错误代码的字符串描述。
curl_unescape()	解码URL编码后的字符串。
curl_version()	获取cURL版本信息。

PHP curl_close函数

(PHP 4 >= 4.0.2, PHP 5)

curl_close — 关闭一个cURL会话

说明

```
void curl_close ( resource $ch )
```

关闭一个cURL会话并且释放所有资源。cURL句柄ch 也会被释放。

参数

ch

由 curl_init() 返回的 cURL 句柄。

返回值

没有返回值。

实例

初始化一个cURL会话来获取一个网页

```
<?php
// 创建一个新cURL资源
$ch = curl_init();

// 设置URL和相应的选项
curl_setopt($ch, CURLOPT_URL, "http://www.w3cschool.cc/");
curl_setopt($ch, CURLOPT_HEADER, 0);

// 抓取URL并把它传递给浏览器
curl_exec($ch);

// 关闭cURL资源，并且释放系统资源
curl_close($ch);
?>
```

参见

- curl_init() - 初始化一个cURL会话
- curl_multi_close() - 关闭一组cURL句柄

PHP curl_copy_handle函数

(PHP 5)

curl_copy_handle — 复制一个cURL句柄和它的所有选项

说明

```
resource curl_copy_handle ( resource $ch )
```

复制一个cURL句柄并保持相同的选项。

参数

ch

由 curl_init() 返回的 cURL 句柄。

返回值

返回一个新的cURL句柄。

实例

复制一个cURL句柄

```
<?php
// 创建一个新的cURL资源
$ch = curl_init();

// 设置URL和相应的选项
curl_setopt($ch, CURLOPT_URL, 'http://www.example.com/');
curl_setopt($ch, CURLOPT_HEADER, 0);

// 复制句柄
$ch2 = curl_copy_handle($ch);

// 抓取URL (http://www.example.com/) 并把它传递给浏览器
curl_exec($ch2);

// 关闭cURL资源，并且释放系统资源
curl_close($ch2);
curl_close($ch);
?>
```

PHP curl_errno函数

(PHP 4 >= 4.0.3, PHP 5)

curl_errno — 返回最后一次的错误号

说明

```
int curl_errno ( resource $ch )
```

返回最后一次cURL操作的错误号。

参数

ch

由 curl_init() 返回的 cURL 句柄。

返回值

返回错误号或 0 (零) 如果没有错误发生。

实例

```
<?php
// 创建一个指向一个不存在的位置的cURL句柄
$ch = curl_init('http://404.php.net/');

// 执行
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_exec($ch);

// 检查是否有错误发生
if(curl_errno($ch))
{
    echo 'Curl error: ' . curl_error($ch);
}

// 关闭句柄
curl_close($ch);
?>
```

参见

- curl_error() - 返回一个保护当前会话最近一次错误的字符串
- ? Curl 错误号

PHP curl_error函数

(PHP 4 >= 4.0.3, PHP 5)

curl_error — 返回一个保护当前会话最近一次错误的字符串

说明

```
string curl_error ( resource $ch )
```

返回一条最近一次cURL操作明确的文本的错误信息。

参数

ch

由 curl_init() 返回的 cURL 句柄。

返回值

返回错误信息或 '' (空字符串) 如果没有任何错误发生。

实例

```
<?php
// 创建一个指向一个不存在的位置的cURL句柄
$ch = curl_init('http://404.php.net/');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

if(curl_exec($ch) === false)
{
    echo 'Curl error: ' . curl_error($ch);
}
else
{
    echo '操作完成没有任何错误';
}

// 关闭句柄
curl_close($ch);
?>
```

参见

- curl_errno() - 返回最后一次的错误号
- ? Curl 错误代码

PHP curl_escape函数

(PHP 5 >= 5.5.0)

curl_escape — 对给定的字符串进行URL编码。

说明

```
string curl_escape ( resource $ch , string $str )
```

该函数对给定的字符串进行URL编码? RFC 3986。

参数

ch

由 curl_init() 返回的 cURL 句柄。

str

编码字符串

返回值

返回编码字符串，或者在失败时返回 FALSE。

实例

```
<?php
// 创建一个cURL句柄
$ch = curl_init();

// 编码GET参数
$location = curl_escape($ch, 'Hofbr?uhaus / München');
// Result: Hofbr%C3%A4uhaus%20%2F%20M%C3%BCnchen

// 比较编码后的URL
$url = "http://example.com/add_location.php?location={$location}";
// Result: http://example.com/add_location.php?location=Hofbr%C3%A4uhaus%20%2F%20M%C3%BCn

// 发送HTTP请求并关闭句柄
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_exec($ch);
curl_close($ch);
?>
```

PHP curl_exec函数

(PHP 4 >= 4.0.2, PHP 5)

curl_exec — 执行一个cURL会话

说明

```
mixed curl_exec ( resource $ch )
```

执行给定的cURL会话。

这个函数应该在初始化一个cURL会话并且全部的选项都被设置后被调用。

参数

ch

由 curl_init() 返回的 cURL 句柄。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。 然而， 如果 CURLOPT_RETURNTRANSFER选项被设置，函数执行成功时会返回执行的结果，失败时返回 FALSE 。

实例

获取一个网页

```
<?php
// 创建一个cURL资源
$ch = curl_init();

// 设置URL和相应的选项
curl_setopt($ch, CURLOPT_URL, "http://www.w3cschool.cc/");
curl_setopt($ch, CURLOPT_HEADER, 0);

// 抓取URL并把它传递给浏览器
curl_exec($ch);

// 关闭cURL资源，并且释放系统资源
curl_close($ch);
?>
```

PHP curl_file_create函数

(PHP 5 >= 5.5.0)

curl_file_create — 创建一个 CURLFile 对象。

说明

```
CURLFile curl_file_create ( string $filename [, string $mimetype [, string $postname ]] )
```

创建一个 CURLFile 对象, 用与上传文件。

参数

filename

上传文件的路径

mimetype

文件的Mimetype

postname

文件名。

返回值

返回 CURLFile 对象。

实例

curl_file_create() 实例

```
<?php
/* http://example.com/upload.php:
<?php var_dump($_FILES); ?>
*/

// 创建一个 cURL 句柄
$ch = curl_init('http://example.com/upload.php');

// 创建一个 CURLFile 对象
$cfile = curl_file_create('cats.jpg','image/jpeg','test_name');

// 设置 POST 数据
$data = array('test_file' => $cfile);
curl_setopt($ch, CURLOPT_POST,1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);

// 执行句柄
curl_exec($ch);
?>
```

以上例程会输出：

```
array(1) {
  ["test_file"]=>
  array(5) {
    ["name"]=>
    string(9) "test_name"
    ["type"]=>
    string(10) "image/jpeg"
    ["tmp_name"]=>
    string(14) "/tmp/phpPC9Kbx"
    ["error"]=>
    int(0)
    ["size"]=>
    int(46334)
  }
}
```

PHP curl_getinfo函数

(PHP 4 >= 4.0.4, PHP 5)

curl_getinfo — 获取一个cURL连接资源句柄的信息

说明

```
mixed curl_getinfo ( resource $ch [, int $opt = 0 ] )
```

获取最后一次传输的相关信息。

参数

ch

由 curl_init() 返回的 cURL 句柄。

opt

这个参数可能是以下常量之一:

- **CURLINFO_EFFECTIVE_URL** - 最后一个有效的URL地址
- **CURLINFO_HTTP_CODE** - 最后一个收到的HTTP代码
- **CURLINFO_FILETIME** - 远程获取文档的时间，如果无法获取，则返回值为"-1"
- **CURLINFO_TOTAL_TIME** - 最后一次传输所消耗的时间
- **CURLINFO_NAMELOOKUP_TIME** - 名称解析所消耗的时间
- **CURLINFO_CONNECT_TIME** - 建立连接所消耗的时间
- **CURLINFO_PRETRANSFER_TIME** - 从建立连接到准备传输所使用的时间
- **CURLINFO_STARTTRANSFER_TIME** - 从建立连接到传输开始所使用的时间
- **CURLINFO_REDIRECT_TIME** - 在事务传输开始前重定向所使用的时间
- **CURLINFO_SIZE_UPLOAD** - 上传数据量的总值
- **CURLINFO_SIZE_DOWNLOAD** - 下载数据量的总值
- **CURLINFO_SPEED_DOWNLOAD** - 平均下载速度
- **CURLINFO_SPEED_UPLOAD** - 平均上传速度
- **CURLINFO_HEADER_SIZE** - header部分的大小
- **CURLINFO_HEADER_OUT** - 发送请求的字符串
- **CURLINFO_REQUEST_SIZE** - 在HTTP请求中有问题的请求的大小
- **CURLINFO_SSL_VERIFYRESULT** - 通过设置**CURLOPT_SSL_VERIFYPEER**返回的SSL证书验证请求的结果

- **CURLINFO_CONTENT_LENGTH_DOWNLOAD** - 从*Content-Length:* field中读取的下载内容长度
- **CURLINFO_CONTENT_LENGTH_UPLOAD** - 上传内容大小的说明
- **CURLINFO_CONTENT_TYPE** - 下载内容的*Content-Type:*值，NULL表示服务器没有发送有效的*Content-Type:* header

返回值

如果 opt 被设置，以字符串形式返回它的值。否则，返回返回一个包含下列元素的关联数组(它们分别对应于 opt):

- "url"
- "content_type"
- "http_code"
- "header_size"
- "request_size"
- "filetime"
- "ssl_verify_result"
- "redirect_count"
- "total_time"
- "namelookup_time"
- "connect_time"
- "pretransfer_time"
- "size_upload"
- "size_download"
- "speed_download"
- "speed_upload"
- "download_content_length"
- "upload_content_length"
- "starttransfer_time"
- "redirect_time"

更新日志

版本	说明
5.1.3	引入 `CURLINFO_HEADER_OUT` .

实例

```
<?php
// 创建一个cURL句柄
$ch = curl_init('http://www.yahoo.com/');

// 执行
curl_exec($ch);

// 检查是否有错误发生
if(!curl_errno($ch))
{
 $info = curl_getinfo($ch);

 echo 'Took ' . $info['total_time'] . ' seconds to send a request to ' . $info['url'];
}

// Close handle
curl_close($ch);
?>
```

PHP curl_init函数

(PHP 4 >= 4.0.2, PHP 5)

curl_init — 初始化一个cURL会话

说明

```
resource curl_init ([ string $url = NULL ] )
```

初始化一个新的会话，返回一个cURL句柄， 供curl_setopt(), curl_exec()和curl_close() 函数使用。

参数

url

如果提供了该参数，CURLOPT_URL 选项将会被设置成这个值。你也可以使用curl_setopt() 函数手动地设置这个值。

返回值

如果成功，返回一个cURL句柄，出错返回 FALSE。

实例

初始化一个新的cURL会话并获取一个网页

```
<?php
// 创建一个新cURL资源
$ch = curl_init();

// 设置URL和相应的选项
curl_setopt($ch, CURLOPT_URL, "http://www.w3cschool.cc/");
curl_setopt($ch, CURLOPT_HEADER, 0);

// 抓取URL并把它传递给浏览器
curl_exec($ch);

// 关闭cURL资源，并且释放系统资源
curl_close($ch);
?>
```

PHP curl_multi_add_handle函数

(PHP 5)

curl_multi_add_handle — 向curl批处理会话中添加单独的curl句柄

说明

```
int curl_multi_add_handle ( resource $mh , resource $ch )
```

增加 ch 句柄到批处理会话mh

参数

mh

由 curl_multi_init() 返回的 cURL 多个句柄。

ch

由 curl_init() 返回的 cURL 句柄。

返回值

成功时返回0，失败时返回CURLM_XXX之一的错误码。

实例

这个范例将会创建2个cURL句柄，把它们加到批处理句柄，然后并行地运行它们。

```
<?php
// 创建一对cURL资源
$ch1 = curl_init();
$ch2 = curl_init();

// 设置URL和相应的选项
curl_setopt($ch1, CURLOPT_URL, "http://www.w3cschool.cc/");
curl_setopt($ch1, CURLOPT_HEADER, 0);
curl_setopt($ch2, CURLOPT_URL, "http://www.php.net/");
curl_setopt($ch2, CURLOPT_HEADER, 0);

// 创建批处理cURL句柄
$mh = curl_multi_init();

// 增加2个句柄
curl_multi_add_handle($mh,$ch1);
curl_multi_add_handle($mh,$ch2);

$running=null;
// 执行批处理句柄
do {
    curl_multi_exec($mh,$running);
} while($running > 0);

// 关闭全部句柄
curl_multi_remove_handle($mh, $ch1);
curl_multi_remove_handle($mh, $ch2);
curl_multi_close($mh);
?>
```

PHP curl_multi_close函数

(PHP 5)

curl_multi_close — 关闭一组cURL句柄

说明

```
void curl_multi_close ( resource $mh )
```

关闭一组cURL句柄。

参数

mh

由 curl_multi_init() 返回的 cURL 多个句柄。

返回值

没有返回值。

实例

这个范例将会创建2个cURL句柄，把它们加到批处理句柄，然后并行地运行它们。

```
<?php
// 创建一对cURL资源
$ch1 = curl_init();
$ch2 = curl_init();

// 设置URL和相应的选项
curl_setopt($ch1, CURLOPT_URL, "http://www.example.com/");
curl_setopt($ch1, CURLOPT_HEADER, 0);
curl_setopt($ch2, CURLOPT_URL, "http://www.php.net/");
curl_setopt($ch2, CURLOPT_HEADER, 0);

// 创建批处理cURL句柄
$mh = curl_multi_init();

// 增加2个句柄
curl_multi_add_handle($mh,$ch1);
curl_multi_add_handle($mh,$ch2);

$running=null;
// 执行批处理句柄
do {
    curl_multi_exec($mh,$running);
} while ($running > 0);

// 关闭全部句柄
curl_multi_remove_handle($mh, $ch1);
curl_multi_remove_handle($mh, $ch2);
curl_multi_close($mh);

?>
```

PHP curl_multi_exec函数

(PHP 5)

curl_multi_exec — 运行当前 cURL 句柄的子连接

说明

```
int curl_multi_exec ( resource $mh , int &$still_running )
```

处理在栈中的每一个句柄。无论该句柄需要读取或写入数据都可调用此方法。

参数

mh

由 curl_multi_init() 返回的 cURL 多个句柄。

still_running

一个用来判断操作是否仍在执行的标识的引用。

返回值

一个定义于 cURL 预定义常量中的 cURL 代码。

注意： 该函数仅返回关于整个批处理栈相关的错误。即使返回 CURLM_OK 时单个传输仍可能有问题。

实例

这个范例将会创建 2 个 cURL 句柄，把它们加到批处理句柄，然后并行地运行它们。

```
<?php
// 创建一对cURL资源
$ch1 = curl_init();
$ch2 = curl_init();

// 设置URL和相应的选项
curl_setopt($ch1, CURLOPT_URL, "http://lxr.php.net/");
curl_setopt($ch1, CURLOPT_HEADER, 0);
curl_setopt($ch2, CURLOPT_URL, "http://www.php.net/");
curl_setopt($ch2, CURLOPT_HEADER, 0);

// 创建批处理cURL句柄
$mh = curl_multi_init();

// 增加2个句柄
curl_multi_add_handle($mh,$ch1);
curl_multi_add_handle($mh,$ch2);

$active = null;
// 执行批处理句柄
do {
    $mrc = curl_multi_exec($mh, $active);
} while ($mrc == CURLM_CALL_MULTI_PERFORM);

while ($active && $mrc == CURLM_OK) {
    if (curl_multi_select($mh) != -1) {
        do {
            $mrc = curl_multi_exec($mh, $active);
        } while ($mrc == CURLM_CALL_MULTI_PERFORM);
    }
}

// 关闭全部句柄
curl_multi_remove_handle($mh, $ch1);
curl_multi_remove_handle($mh, $ch2);
curl_multi_close($mh);

?>
```

PHP curl_multi_getcontent函数

(PHP 5)

curl_multi_getcontent — 如果设置了CURLOPT_RETURNTRANSFER，则返回获取的输出的文本流

说明

```
string curl_multi_getcontent ( resource $ch )
```

如果CURLOPT_RETURNTRANSFER作为一个选项被设置到一个具体的句柄，那么这个函数将会以字符串的形式返回那个cURL句柄获取的内容。

参数

mh

由 curl_multi_init() 返回的 cURL 多个句柄。

返回值

如果设置了CURLOPT_RETURNTRANSFER，则返回获取的输出的文本流。

PHP curl_multi_info_read函数

(PHP 5)

curl_multi_info_read — 获取当前解析的cURL的相关传输信息

说明

```
array curl_multi_info_read ( resource $mh [, int &$msgs_in_queue = NULL ] )
```

查询批处理句柄是否单独的传输线程中有消息或信息返回。消息可能包含诸如从单独的传输线程返回的错误码或者只是传输线程有没有完成之类的报告。

重复调用这个函数，它每次都会返回一个新的结果，直到这时没有更多信息返回时，FALSE被当作一个信号返回。通过msgs_in_queue返回的整数指出将会包含当这次函数被调用后，还剩余的消息数。

注意: 返回的资源指向的数据调用curl_multi_remove_handle()后将不会存在。

参数

mh

由 curl_multi_init() 返回的 cURL 多个句柄。

msgs_in_queue

仍在队列中的消息数量。

返回值

成功时返回相关信息的数组，失败时返回FALSE。

返回值内容（返回数组的内容）：

键	值
msg	`CURLMSG_DONE` 常量。其他返回值当前不可用。
result	`CURLE_*` 常量之一。如果一切操作没有问题，将会返回 `CURLE_OK` 常量。
handle	cURL资源类型表明它有关的句柄。

实例

```
<?php

$urls = array(
   "http://www.baidu.com/",
   "http://www.google.com.hk/",
   "http://www.w3cschool.cc/"
);

$mh = curl_multi_init();

foreach ($urls as $i => $url) {
    $conn[$i] = curl_init($url);
    curl_setopt($conn[$i], CURLOPT_RETURNTRANSFER, 1);
    curl_multi_add_handle($mh, $conn[$i]);
}

do {
    $status = curl_multi_exec($mh, $active);
    $info = curl_multi_info_read($mh);
    if (false !== $info) {
        var_dump($info);
    }
} while ($status === CURLM_CALL_MULTI_PERFORM || $active);

foreach ($urls as $i => $url) {
    $res[$i] = curl_multi_getcontent($conn[$i]);
    curl_close($conn[$i]);
}

var_dump(curl_multi_info_read($mh));

?>
```

以上例程的输出类似于：

```
array(3) {
  ["msg"]=>
  int(1)
  ["result"]=>
  int(0)
  ["handle"]=>
  resource(5) of type (curl)
}
array(3) {
  ["msg"]=>
  int(1)
  ["result"]=>
  int(0)
  ["handle"]=>
  resource(7) of type (curl)
}
array(3) {
  ["msg"]=>
  int(1)
  ["result"]=>
  int(0)
  ["handle"]=>
  resource(6) of type (curl)
}
bool(false)
```

更新日志

版本	说明
5.2.0	`msgs_in_queue` 被加入。

PHP curl_multi_init函数

(PHP 5)

curl_multi_init — 返回一个新cURL批处理句柄

说明

```
resource curl_multi_init ( void )
```

允许并行地处理批处理cURL句柄。

参数

此函数没有参数。

返回值

成功时返回一个cURL批处理句柄，失败时返回FALSE。

实例

这个范例将会创建2个cURL句柄，把它们加到批处理句柄，然后并行地运行它们。

```
<?php
// 创建一对cURL资源
$ch1 = curl_init();
$ch2 = curl_init();

// 设置URL和相应的选项
curl_setopt($ch1, CURLOPT_URL, "http://www.example.com/");
curl_setopt($ch1, CURLOPT_HEADER, 0);
curl_setopt($ch2, CURLOPT_URL, "http://www.php.net/");
curl_setopt($ch2, CURLOPT_HEADER, 0);

// 创建批处理cURL句柄
$mh = curl_multi_init();

// 增加2个句柄
curl_multi_add_handle($mh,$ch1);
curl_multi_add_handle($mh,$ch2);

$running=null;
// 执行批处理句柄
do {
    usleep(10000);
    curl_multi_exec($mh,$running);
} while ($running > 0);

// 关闭全部句柄
curl_multi_remove_handle($mh, $ch1);
curl_multi_remove_handle($mh, $ch2);
curl_multi_close($mh);

?>
```

PHP curl_multi_remove_handle函数

(PHP 5)

curl_multi_remove_handle — 移除curl批处理句柄资源中的某个句柄资源

说明

```
int curl_multi_remove_handle ( resource $mh , resource $ch )
```

从给定的批处理句柄mh中移除ch句柄。当ch句柄被移除以后，仍然可以合法地用curl_exec()执行这个句柄。当正在移除的句柄正在被使用，在处理的过程中所有的传输任务会被终止。

参数

mh

由 curl_multi_init() 返回的 cURL 多个句柄。

ch

由 curl_init() 返回的 cURL 句柄。

返回值

成功时返回一个cURL句柄，失败时返回FALSE。

PHP curl_multi_select函数

(PHP 5)

curl_multi_select — 等待所有cURL批处理中的活动连接

说明

```
int curl_multi_select ( resource $mh [, float $timeout = 1.0 ] )
```

阻塞直到cURL批处理连接中有活动连接。

参数

mh

由 curl_multi_init() 返回的 cURL 多个句柄。

timeout

以秒为单位，等待响应的时间。

返回值

成功时返回描述符集合中描述符的数量。失败时，select失败时返回-1，否则返回超时(从底层的select系统调用).

PHP curl_multi_setopt函数

(PHP 5 >= 5.5.0)

curl_multi_setopt — 设置一个批处理cURL传输选项。

说明

```
bool curl_multi_setopt ( resource $mh , int $option , mixed $value )
```

设置一个批处理cURL传输选项。

参数

ch

由 curl_init() 返回的 cURL 句柄。

option

需要设置的CURLOPT_XXX选项。

value

将设置在option选项上的值。

对于下面的这些option的可选参数，value应该被设置一个bool类型的值：

选项	可选***value***值
CURLOPT_AUTOREFERER	当根据*Location:*重定向时，自动设置header中 *Referer:*信息。
CURLOPT_BINARYTRANSFER	在启用**CURLOPT_RETURNTRANSFER**的时 返回原生的（Raw）输出。
CURLOPT_COOKIESESSION	启用时curl会仅仅传递一个session cookie，忽 他的cookie，默认状况下cURL会将所有的co 回给服务端。session cookie是指那些用来判 务器端的session是否有效而存在的cookie。
CURLOPT_CRLF	启用时将Unix的换行符转换成回车换行符。
CURLOPT_DNS_USE_GLOBAL_CACHE	启用时会启用一个全局的DNS缓存，此项为线 全的，并且默认启用。
	显示HTTP状态码，默认行为是忽略编号小于

CURLOPT_FAILONERROR	400的HTTP信息。
CURLOPT_FILETIME	启用时会尝试修改远程文档中的信息。结果信通过curl*getinfo()*函数的*_CURLINFO_FILETI*项返回。 curl_getinfo().
CURLOPT_FOLLOWLOCATION	启用时会将服务器服务器返回的*"Location: "*header中递归的返回给服务器，使用**CURLOPT_MAXREDIRS**可以限定递归返数量。
CURLOPT_FORBID_REUSE	在完成交互以后强迫断开连接，不能重用。
CURLOPT_FRESH_CONNECT	强制获取一个新的连接，替代缓存中的连接。
CURLOPT_FTP_USE_EPRT	启用时当FTP下载时，使用EPRT (或 LPRT)设置为**FALSE**时禁用EPRT和LPRT，使用PC令 only.
CURLOPT_FTP_USE_EPSV	启用时，在FTP传输过程中回复到PASV模式先尝试EPSV命令。设置为**FALSE**时禁用EPS令。
CURLOPT_FTPAPPEND	启用时追加写入文件而不是覆盖它。
CURLOPT_FTPASCII	**CURLOPT_TRANSFERTEXT**的别名。
CURLOPT_FTPLISTONLY	启用时只列出FTP目录的名字。
CURLOPT_HEADER	启用时会将头文件的信息作为数据流输出。
CURLINFO_HEADER_OUT	启用时追踪句柄的请求字符串。
CURLOPT_HTTPGET	启用时会设置HTTP的method为GET，因为G默认是，所以只在被修改的情况下使用。
CURLOPT_HTTPPROXYTUNNEL	启用时会通过HTTP代理来传输。
CURLOPT_MUTE	启用时将cURL函数中所有修改过的参数恢复值。
CURLOPT_NETRC	在连接建立以后，访问<var class="filename">~/.netrc</var>文件获取用户密码信息连接远程站点。
CURLOPT_NOBODY	启用时将不对HTML中的BODY部分进行输出
CURLOPT_NOPROGRESS	启用时关闭curl传输的进度条，此项的默认设启用。**Note**: PHP自动地设置这个选项为**TRU**这个选项仅仅应当在以调试为目的时被改变。
CURLOPT_NOSIGNAL	启用时忽略所有的curl传递给php进行的信号SAPI多线程传输时此项被默认启用。
CURLOPT_POST	启用时会发送一个常规的POST请求，类型为：*application/x-www-form-urlencoded*，就

	单提交的一样。
CURLOPT_PUT	启用时允许HTTP发送文件，必须同时设置**CURLOPT_INFILE**和**CURLOPT_INFILES**
CURLOPT_RETURNTRANSFER	将curl_exec()获取的信息以文件流的形式返回不是直接输出。
CURLOPT_SSL_VERIFYPEER	禁用后cURL将终止从服务端进行验证。使用**CURLOPT_CAINFO**选项设置证书使用**CURLOPT_CAPATH**选项设置证书目录 如**CURLOPT_SSL_VERIFYPEER**(默认值为2)用，**CURLOPT_SSL_VERIFYHOST**需要被成**TRUE**否则设置为**FALSE**。
CURLOPT_TRANSFERTEXT	启用后对FTP传输使用ASCII模式。对于LDA检索纯文本信息而非HTML。在Windows系统系统不会把*STDOUT*设置成binary模式。
CURLOPT_UNRESTRICTED_AUTH	在使用**CURLOPT_FOLLOWLOCATION**产生header中的多个locations中持续追加用户名信息，即使域名已发生改变。
CURLOPT_UPLOAD	启用后允许文件上传。
CURLOPT_VERBOSE	启用时会汇报所有的信息，存放在*STDERR*的**CURLOPT_STDERR**中。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

PHP curl_multi_strerror函数

(PHP 5 >= 5.5.0)

curl_multi_setopt — 返回描述错误码的字符串文本。

说明

```
string curl_multi_strerror ( int $errornum )
```

返回描述 CURLM 错误码的字符串文本。

参数

errornum

一个? CURLM error codes错误码常量。

返回值

返回描述错误码的字符串文本, 否则返回 NULL。

实例

```
<?php
// 创建 cURL 句柄
$ch1 = curl_init("http://www.w3cschool.cc/");
$ch2 = curl_init("http://php.net/");

// 创建一个批处理cURL句柄
$mh = curl_multi_init();

// 添加句柄到批处理句柄
curl_multi_add_handle($mh, $ch1);
curl_multi_add_handle($mh, $ch2);

// 执行批处理句柄
do {
    $status = curl_multi_exec($mh, $active);
    // 检查错误
    if($status > 0) {
        // 显示错误信息
        echo "ERROR!\n " . curl_multi_strerror($status);
    }
} while ($status === CURLM_CALL_MULTI_PERFORM || $active);
?>
```

PHP curl_pause函数

(PHP 5 >= 5.5.0)

curl_pause — 暂停及恢复连接。

说明

```
int curl_pause ( resource $ch , int $bitmask )
```

参数

ch

由 curl_init() 返回的 cURL 句柄。

bitmask

CURLPAUSE_*中的一个常量。

返回值

返回一个错误代码，如果没有错误返回 CURLE_OK 。

PHP curl_reset函数

(PHP 5 >= 5.5.0)

curl_reset— 重置libcurl会话句柄的所有选项。

说明

```
void curl_reset ( resource $ch )
```

该函数将重新初始化cURL的所有选项值（默认值）。

注意：curl_reset() 同样会重新设置 curl_init() 的 URL 参数。

参数

ch

由 curl_init() 返回的 cURL 句柄。

返回值

没有返回值。

实例

```
<?php
// 创建一个cURL句柄
$ch = curl_init();

// 设置 CURLOPT_USERAGENT 选项
curl_setopt($ch, CURLOPT_USERAGENT, "My test user-agent");

// 重置所有先前设置的选项
curl_reset($ch);

// 发送 HTTP 请求
curl_setopt($ch, CURLOPT_URL, 'http://w3cschool.cc/');
curl_exec($ch); // the previously set user-agent will be not sent, it has been reset by c

// 关闭句柄
curl_close($ch);
?>
```

PHP curl_setopt_array函数

(PHP 5 >= 5.1.3)

curl_setopt_array — 为cURL传输会话批量设置选项。

说明

```
bool curl_setopt_array ( resource $ch , array $options )
```

为cURL传输会话批量设置选项。这个函数对于需要设置大量的cURL选项是非常有用的，不需要重复地调用curl_setopt()。

参数

ch

由 curl_init() 返回的 cURL 句柄。

options

一个array用来确定将被设置的选项及其值。数组的键值必须是一个有效的curl_setopt()常量或者是它们对等的整数值。

返回值

如果全部的选项都被成功设置，返回TRUE。如果一个选项不能被成功设置，马上返回FALSE，忽略其后的任何在options数组中的选项。

实例

初始化一个新的cURL辉煌并抓取一个web页面。

```
<?php
// 创建一个新cURL资源
$ch = curl_init();

// 设置URL和相应的选项
$options = array(CURLOPT_URL => 'http://www.w3cschool.cc/',
                 CURLOPT_HEADER => false
                );

curl_setopt_array($ch, $options);

// 抓取URL并把它传递给浏览器
curl_exec($ch);

// 关闭cURL资源，并且释放系统资源
curl_close($ch);
?>
```

早于PHP 5.1.3这个函数可以做如下模拟：

我们对curl_setopt_array()的等价实现

```
<?php
if (!function_exists('curl_setopt_array')) {
   function curl_setopt_array(&$ch, $curl_options)
   {
       foreach ($curl_options as $option => $value) {
           if (!curl_setopt($ch, $option, $value)) {
               return false;
           }
       }
       return true;
   }
}
?>
```

注意：就curl_setopt()来说，传递一个数组到CURLOPT_POST将会把数据以multipart/form-data的方式编码，然而传递一个URL-encoded字符串将会以application/x-www-form-urlencoded的方式对数据进行编码。

PHP curl_setopt函数

(PHP 4 >= 4.0.2, PHP 5)

curl_setopt — 设置一个cURL传输选项。

说明

```
bool curl_setopt ( resource $ch , int $option , mixed $value )
```

为给定的cURL会话句柄设置一个选项。

参数

ch

由 curl_init() 返回的 cURL 句柄。

option

需要设置的CURLOPT_XXX选项。

value

将设置在option选项上的值。

对于下面的这些option的可选参数，value应该被设置一个bool类型的值：

选项	可选*value*值
CURLOPT_AUTOREFERER	当根据*Location:*重定向时，自动设置header中 *Referer:*信息。
CURLOPT_BINARYTRANSFER	在启用**CURLOPT_RETURNTRANSFER**的时 返回原生的（Raw）输出。
CURLOPT_COOKIESESSION	启用时curl会仅仅传递一个session cookie，忽 他的cookie，默认状况下cURL会将所有的co 回给服务端。session cookie是指那些用来判 务器端的session是否有效而存在的cookie。
CURLOPT_CRLF	启用时将Unix的换行符转换成回车换行符。
CURLOPT_DNS_USE_GLOBAL_CACHE	启用时会启用一个全局的DNS缓存，此项为线 全的，并且默认启用。
CURLOPT_FAILONERROR	显示HTTP状态码，默认行为是忽略编号小于

CURLOPT_FAILONERROR	400的HTTP信息。
CURLOPT_FILETIME	启用时会尝试修改远程文档中的信息。结果信通过curl*getinfo()*函数的_*CURLINFO_FILETI*项返回。 curl_getinfo().
CURLOPT_FOLLOWLOCATION	启用时会将服务器服务器返回的*"Location: "*放header中递归的返回给服务器，使用**CURLOPT_MAXREDIRS**可以限定递归返数量。
CURLOPT_FORBID_REUSE	在完成交互以后强迫断开连接，不能重用。
CURLOPT_FRESH_CONNECT	强制获取一个新的连接，替代缓存中的连接。
CURLOPT_FTP_USE_EPRT	启用时当FTP下载时，使用EPRT (或 LPRT)设置为**FALSE**时禁用EPRT和LPRT，使用PO令 only.
CURLOPT_FTP_USE_EPSV	启用时，在FTP传输过程中回复到PASV模式先尝试EPSV命令。设置为**FALSE**时禁用EPS令。
CURLOPT_FTPAPPEND	启用时追加写入文件而不是覆盖它。
CURLOPT_FTPASCII	**CURLOPT_TRANSFERTEXT**的别名。
CURLOPT_FTPLISTONLY	启用时只列出FTP目录的名字。
CURLOPT_HEADER	启用时会将头文件的信息作为数据流输出。
CURLINFO_HEADER_OUT	启用时追踪句柄的请求字符串。
CURLOPT_HTTPGET	启用时会设置HTTP的method为GET，因为G默认是，所以只在被修改的情况下使用。
CURLOPT_HTTPPROXYTUNNEL	启用时会通过HTTP代理来传输。
CURLOPT_MUTE	启用时将cURL函数中所有修改过的参数恢复值。
CURLOPT_NETRC	在连接建立以后，访问<var class="filename">~/.netrc</var>文件获取用户密码信息连接远程站点。
CURLOPT_NOBODY	启用时将不对HTML中的BODY部分进行输出
CURLOPT_NOPROGRESS	启用时关闭curl传输的进度条，此项的默认设启用。**Note**: PHP自动地设置这个选项为**TRU**这个选项仅仅应当在以调试为目的时被改变。
CURLOPT_NOSIGNAL	启用时忽略所有的curl传递给php进行的信号。SAPI多线程传输时此项被默认启用。
	启用时会发送一个常规的POST请求，类型

	单提交的一样。
CURLOPT_PUT	启用时允许HTTP发送文件，必须同时设置**CURLOPT_INFILE**和**CURLOPT_INFILES**
CURLOPT_RETURNTRANSFER	将curl_exec()获取的信息以文件流的形式返回不是直接输出。
CURLOPT_SSL_VERIFYPEER	禁用后cURL将终止从服务端进行验证。使用**CURLOPT_CAINFO**选项设置证书使用**CURLOPT_CAPATH**选项设置证书目录 如**CURLOPT_SSL_VERIFYPEER**(默认值为2)用，**CURLOPT_SSL_VERIFYHOST**需要被成**TRUE**否则设置为**FALSE**。
CURLOPT_TRANSFERTEXT	启用后对FTP传输使用ASCII模式。对于LDA检索纯文本信息而非HTML。在Windows系统不会把*STDOUT*设置成binary模式。
CURLOPT_UNRESTRICTED_AUTH	在使用**CURLOPT_FOLLOWLOCATION**产生header中的多个locations中持续追加用户名信息，即使域名已发生改变。
CURLOPT_UPLOAD	启用后允许文件上传。
CURLOPT_VERBOSE	启用时会汇报所有的信息，存放在*STDERR*的**CURLOPT_STDERR**中。

对于下面的这些option的可选参数，value应该被设置一个integer类型的值：

选项	可选*value*值
CURLOPT_BUFFERSIZE	每次获取的数据中读入缓存的大小，但是不保证这个值每次都会被填满。
CURLOPT_CLOSEPOLICY	不是CURLCLOSEPOLICY_LEAST_RECENTLY_USI就是CURLCLOSEPOLICY_OLDEST，还存在另三个CURLCLOSEPOLICY，但是cURL暂时还不持。
CURLOPT_CONNECTTIMEOUT	在发起连接前等待的时间，如果设置为0，则无限待。
CURLOPT_CONNECTTIMEOUT_MS	尝试连接等待的时间，以毫秒为单位。如果设置0，则无限等待。

CURLOPT_DNS_CACHE_TIMEOUT	设置在内存中保存DNS信息的时间，默认为120秒。
CURLOPT_FTPSSLAUTH	FTP验证方式：*CURLFTPAUTH_SSL* (首先尝试SSL)，*CURLFTPAUTH_TLS* (首先尝试TLS)或*CURLFTPAUTH_DEFAULT* (让cURL自动决定)
CURLOPT_HTTP_VERSION	*CURL_HTTP_VERSION_NONE* (默认值，让cUR自己判断使用哪个版本)，*CURL_HTTP_VERSION_1_0* (强制使用HTTP/1.0)或*CURL_HTTP_VERSION_1_1* (强制使用 HTTP/1.1)。
CURLOPT_INFILESIZE	设定上传文件的大小限制，字节(byte)为单位。
CURLOPT_LOW_SPEED_LIMIT	当传输速度小于**CURLOPT_LOW_SPEED_LIMIT**时(bytes/sec)，PHP会根据**CURLOPT_LOW_SPEED_TIME**来判断是否因慢而取消传输。
CURLOPT_LOW_SPEED_TIME	当传输速度小于**CURLOPT_LOW_SPEED_LIMIT**时(bytes/sec)，PHP会根据**CURLOPT_LOW_SPEED_TIME**来判断是否因慢而取消传输。
CURLOPT_MAXCONNECTS	允许的最大连接数量，超过是会通过**CURLOPT_CLOSEPOLICY**决定应该停止哪些连接。
CURLOPT_MAXREDIRS	指定最多的HTTP重定向的数量，这个选项是和**CURLOPT_FOLLOWLOCATION**一起使用的。
CURLOPT_PORT	用来指定连接端口。（可选项）
CURLOPT_PROTOCOLS	**CURLPROTO_***的位域指。如果被启用，位域值会限定libcurl在传输过程中有哪些可使用的协议。这允许你在编译libcurl时支持众多协议，但是限制只用它们中被允许使用的一个子集。默认libcurl将会用全部它支持的协议。参见**CURLOPT_REDIR_PROTOCOLS**.可用的协议选为：CURLPROTO_HTTP、CURLPROTO_HTTPS、CURLPROTO_FTP、CURLPROTO_FTPS、CURLPROTO_SCP、CURLPROTO_SFTP、CURLPROTO_TELNET、CURLPROTO_LDAP、CURLPROTO_LDAPS、CURLPROTO_DICT、CURLPROTO_FILE、CURLPROTO_TFTP、CURLPROTO_ALL

CURLOPT_PROTOCOLS	**CURLPROTO_***的位域指。如果被启用，位域值会限定libcurl在传输过程中有哪些可使用的协议。这允许你在编译libcurl时支持众多协议，但是限制只用它们中被允许使用的一个子集。默认libcurl将会用全部它支持的协议。参见**CURLOPT_REDIR_PROTOCOLS**.可用的协议选为：CURLPROTO_HTTP、CURLPROTO_HTTPS、CURLPROTO_FTP、CURLPROTO_FTPS、CURLPROTO_SCP、CURLPROTO_SFTP、CURLPROTO_TELNET、CURLPROTO_LDAP、CURLPROTO_LDAPS、CURLPROTO_DICT、CURLPROTO_FILE、CURLPROTO_TFTP、CURLPROTO_ALL
CURLOPT_PROXYAUTH	HTTP代理连接的验证方式。使用在**CURLOPT_HTTPAUTH**中的位域标志来设置相应选项。对于代理验证只有*CURLAUTH_BASIC*和*CURLAUTH_NTLM*当前被支持。
CURLOPT_PROXYPORT	代理服务器的端口。端口也可以在**CURLOPT_PROXY**中进行设置。
CURLOPT_PROXYTYPE	不是*CURLPROXY_HTTP* (默认值) 就是*CURLPROXY_SOCKS5*。
CURLOPT_REDIR_PROTOCOLS	**CURLPROTO_***中的位域值。如果被启用，位域值将会限制传输线程在**CURLOPT_FOLLOWLOCATION**开启时跟随某个重定向时可使用的协议。这将使你对重定向时限制传输线程使用被允许的协议子集默认libcurl将会允许除FILE和SCP之外的全部协议。这个和7.19.4预发布版本种无条件地跟随所有支持的协议有一些不同。关于协议常量，请参照**CURLOPT_PROTOCOLS**。
CURLOPT_RESUME_FROM	在恢复传输时传递一个字节偏移量（用来断点续传）。
CURLOPT_SSL_VERIFYHOST	1 检查服务器SSL证书中是否存在一个公用名(common name)。译者注：公用名(Common Name)一般来讲就是填写你将要申请SSL证书的域名 (domain)或子域名(sub domain)。2 检查公用名是否存在，并且是否与提供的主机名匹配。
CURLOPT_SSLVERSION	使用的SSL版本(2 或 3)。默认情况下PHP会自己检测这个值，尽管有些情况下需要手动地进行设置。
	如果在**CURLOPT_TIMEVALUE**指定的某个时间以后被编辑过，则使

CURLOPT_TIMECONDITION	用*CURL_TIMECOND_IFMODSINCE*返回页面，果没有被修改过，并且**CURLOPT_HEADER**为true，则返回一个*"304 Not Modified"*的header，**CURLOPT_HEADER**为false，则使用*CURL_TIMECOND_IFUNMODSINCE*，默认值为*CURL_TIMECOND_IFUNMODSINCE*。
CURLOPT_TIMEOUT	设置cURL允许执行的最长秒数。
CURLOPT_TIMEOUT_MS	设置cURL允许执行的最长毫秒数。
CURLOPT_TIMEVALUE	设置一个**CURLOPT_TIMECONDITION**使用的时戳，在默认状态下使用的是*CURL_TIMECOND_IFMODSINCE*。

对于下面的这些option的可选参数，value应该被设置一个string类型的值：

选项	可选*value*值
CURLOPT_CAINFO	一个保存着1个或多个用来让服务端验证的证书的文件名。这个参数仅仅在和**CURLOPT_SSL_VERIFYPEER**一起使用时才有意义。 .
CURLOPT_CAPATH	一个保存着多个CA证书的目录。这个选项是和**CURLOPT_SSL_VERIFYPEER**一起使用的。
CURLOPT_COOKIE	设定HTTP请求中*"Cookie: "*部分的内容。多个cookie用分号分隔，分号后带一个空格(例如，"*fruit=apple; colour=red*")。
CURLOPT_COOKIEFILE	包含cookie数据的文件名，cookie文件的格式可以是Netscape格式，或者只是纯HTTP头部信息存入文件。
CURLOPT_COOKIEJAR	连接结束后保存cookie信息的文件。
CURLOPT_CUSTOMREQUEST	使用一个自定义的请求信息来代替*"GET"*或*"HEAD"*作为HTTP请求。这对于执行*"DELETE"* 或者其他更隐蔽的HTTP请求。有效值如*"GET"*，*"POST"*，*"CONNECT"*等等。也就是说，不要在这里输入整个HTTP请求。例如输入*"GET /index.html HTTP/1.0\r\n\r\n"*是不正确的。**Note**: 在确定服务器支持这个自定义请求的

	方法前不要使用。
CURLOPT_EGDSOCKET	类似**CURLOPT_RANDOM_FILE**，除了一个Entropy Gathering Daemon套接字。
CURLOPT_ENCODING	HTTP请求头中*"Accept-Encoding: "*的值。支持的编码有*"identity"*，*"deflate"*和*"gzip"*。如果为空字符串*""*，请求头会发送所有支持的编码类型。
CURLOPT_FTPPORT	这个值将被用来获取供FTP"POST"指令所需要的IP地址。"POST"指令告诉远程服务器连接到我们指定的IP地址。这个字符串可以是纯文本的IP地址、主机名、一个网络接口名（UNIX下）或者只是一个'-'来使用默认的IP地址。
CURLOPT_INTERFACE	网络发送接口名，可以是一个接口名、IP地址或者是一个主机名。
CURLOPT_KRB4LEVEL	KRB4 (Kerberos 4) 安全级别。下面的任何值都是有效的(从低到高的顺序)：*"clear"*、*"safe"*、*"confidential"*、*"private"*.。如果字符串和这些都不匹配，将使用*"private"*。这个选项设置为**NULL**时将禁用KRB4 安全认证。目前KRB4 安全认证只能用于FTP传输。
CURLOPT_POSTFIELDS	全部数据使用HTTP协议中的"POST"操作来发送。要发送文件，在文件名前面加上@前缀并使用完整路径。这个参数可以通过urlencoded后的字符串类似'*para1=val1&para2=val2&...*'或使用一个以字段名为键值，字段数据为值的数组。如果*value*是一个数组，*Content-Type*头将会被设置成*multipart/form-data*。
CURLOPT_PROXY	HTTP代理通道。
CURLOPT_PROXYUSERPWD	一个用来连接到代理的*"[username]:[password]"*格式的字符串。
CURLOPT_RANDOM_FILE	一个被用来生成SSL随机数种子的文件名。
CURLOPT_RANGE	以*"X-Y"*的形式，其中X和Y都是可选项获取数据的范围，以字节计。HTTP传输线程也支持几个这样的重复项中间用逗号分隔如*"X-Y,N-M"*。
CURLOPT_REFERER	在HTTP请求头中*"Referer: "*的内容。
CURLOPT_SSL_CIPHER_LIST	一个SSL的加密算法列表。例如*RC4-SHA*和*TLSv1*都是可用的加密列表。
CURLOPT_SSLCERT	一个包含PEM格式证书的文件名。
CURLOPT_SSLCERTPASSWD	使用**CURLOPT_SSLCERT**证书需要的密码。

CURLOPT_SSLCERTTYPE	证书的类型。支持的格式有*"PEM"* (默认值), *"DER"*和*"ENG"*。
CURLOPT_SSLENGINE	用来在**CURLOPT_SSLKEY**中指定的SSL私钥的加密引擎变量。
CURLOPT_SSLENGINE_DEFAULT	用来做非对称加密操作的变量。
CURLOPT_SSLKEY	包含SSL私钥的文件名。
CURLOPT_SSLKEYPASSWD	在**CURLOPT_SSLKEY**中指定了的SSL私钥的密码。**Note**: 由于这个选项包含了敏感的密码信息，记得保证这个PHP脚本的安全。
CURLOPT_SSLKEYTYPE	**CURLOPT_SSLKEY**中规定的私钥的加密类型，支持的密钥类型为*"PEM"*(默认值)、*"DER"*和*"ENG"*。
CURLOPT_URL	需要获取的URL地址，也可以在curl_init()函数中设置。
CURLOPT_USERAGENT	在HTTP请求中包含一个*"User-Agent: "*头的字符串。
CURLOPT_USERPWD	传递一个连接中需要的用户名和密码，格式为：*"[username]:[password]"*。

对于下面的这些option的可选参数，value应该被设置一个数组：

选项	可选***value***值	备注
CURLOPT_HTTP200ALIASES	200响应码数组，数组中的响应吗被认为是正确的响应，否则被认为是错误的。	在cURL 7.10.3中被加入。
CURLOPT_HTTPHEADER	一个用来设置HTTP头字段的数组。使用如下的形式的数组进行设置：array('Content-type: text/plain', 'Content-length: 100')	
CURLOPT_POSTQUOTE	在FTP请求执行完成后，在服务器上执行的一组FTP命令。	
CURLOPT_QUOTE	一组先于FTP请求的在服务器上执行的FTP命令。	

对于下面的这些option的可选参数，value应该被设置一个流资源 （例如使用fopen()）：

选项	可选*value*值
CURLOPT_FILE	设置输出文件的位置，值是一个资源类型，默认为*STDOUT* (浏览器)。
CURLOPT_INFILE	在上传文件的时候需要读取的文件地址，值是一个资源类型。
CURLOPT_STDERR	设置一个错误输出地址，值是一个资源类型，取代默认的*STDERR*。
CURLOPT_WRITEHEADER	设置header部分内容的写入的文件地址，值是一个资源类型。

对于下面的这些option的可选参数，value应该被设置为一个回调函数名：

选项	可选*value*值
CURLOPT_HEADERFUNCTION	设置一个回调函数，这个函数有两个参数，第一个是cURL的资源句柄，第二个是输出的header数据。header数据的输出必须依赖这个函数，返回已写入的数据大小。
CURLOPT_PASSWDFUNCTION	设置一个回调函数，有三个参数，第一个是cURL的资源句柄，第二个是一个密码提示符，第三个参数是密码长度允许的最大值。返回密码的值。
CURLOPT_PROGRESSFUNCTION	设置一个回调函数，有三个参数，第一个是cURL的资源句柄，第二个是一个文件描述符资源，第三个是长度。返回包含的数据。
CURLOPT_READFUNCTION	回调函数名。该函数应接受三个参数。第一个是 cURL resource；第二个是通过选项 **CURLOPT_INFILE** 传给 cURL 的 stream resource；第三个参数是最大可以读取的数据的数量。回 调函数必须返回一个字符串，长度小于或等于请求的数据量（第三个参数）。一般从传入的 stream resource 读取。返回空字符串作为 *EOF*（文件结束） 信号。
CURLOPT_WRITEFUNCTION	回调函数名。该函数应接受两个参数。第一个是 cURL resource；第二个是要写入的数据字符串。数 据必须在函数中被保存。函数必须返回准确的传入的要写入数据的字节数，否则传输会被一个错误所中 断。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

更新日志

版本	说明
5.2.10	引入 `CURLOPT_PROTOCOLS` , and `CURLOPT_REDIR_PROTOCOLS` .
5.1.0	引入 `CURLOPT_AUTOREFERER` , `CURLOPT_BINARYTRANSFER` , `CURLOPT_FTPSSLAUTH` , `CURLOPT_PROXYAUTH` , and `CURLOPT_TIMECONDITION` .
5.0.0	引入 `CURLOPT_FTP_USE_EPRT` , `CURLOPT_NOSIGNAL` , `CURLOPT_UNRESTRICTED_AUTH` , `CURLOPT_BUFFERSIZE` , `CURLOPT_HTTPAUTH` , `CURLOPT_PROXYPORT` , `CURLOPT_PROXYTYPE` , `CURLOPT_SSLCERTTYPE` , and `CURLOPT_HTTP200ALIASES` .

实例

初始化一个新的cURL会话并获取一个网页

```
<?php
// 创建一个新cURL资源
$ch = curl_init();

// 设置URL和相应的选项
curl_setopt($ch, CURLOPT_URL, "http://www.example.com/");
curl_setopt($ch, CURLOPT_HEADER, false);

// 抓取URL并把它传递给浏览器
curl_exec($ch);

//关闭cURL资源，并且释放系统资源
curl_close($ch);
?>
```

上传文件实例:

```
<?php

/* http://localhost/upload.php:
print_r($_POST);
print_r($_FILES);
*/

$ch = curl_init();

$data = array('name' => 'Foo', 'file' => '@/home/user/test.png');

curl_setopt($ch, CURLOPT_URL, 'http://localhost/upload.php');
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $data);

curl_exec($ch);
?>
```

以上实例输出结果如下：

```
Array
(
    [name] => Foo
)
Array
(
    [file] => Array
        (
            [name] => test.png
            [type] => image/png
            [tmp_name] => /tmp/phpcpjNeQ
            [error] => 0
            [size] => 279
        )

)
```

注释

传递一个数组到CURLOPT_POSTFIELDS，cURL会把数据编码成 multipart/form-data，而然传递一个URL-encoded字符串时，数据会被编码成 application/x-www-form-urlencoded。

PHP curl_share_close函数

(PHP 5 >= 5.5.0)

curl_share_close — 关闭 cURL 共享句柄

说明

```
void curl_share_close ( resource $sh )
```

关闭 cURL 共享句柄并释放所有资源。

参数

sh

通过curl_share_init()返回cURL共享句柄。

返回值

没有返回值。

实例

该实例将创建一个cURL共享句柄，并添加两个 cURL 句柄，两个句柄共享cookie数据。

```
<?php
// 创建cURL共享句柄并设置cookie数据
$sh = curl_share_init();
curl_share_setopt($sh, CURLSHOPT_SHARE, CURL_LOCK_DATA_COOKIE);

// 初始化第一个cURL句柄并指定它为共享句柄
$ch1 = curl_init("http://www.w3cschool.cc/");
curl_setopt($ch1, CURLOPT_SHARE, $sh);

// 执行第一个cURL句柄
curl_exec($ch1);

// 初始化第二个cURL句柄并指定它为共享句柄
$ch2 = curl_init("http://php.net/");
curl_setopt($ch2, CURLOPT_SHARE, $sh);

// 执行第二个cURL句柄
//  所有 $ch1 句柄的数据在 $ch2 句柄中共享
curl_exec($ch2);

// 关闭cURL共享句柄
curl_share_close($sh);

// 关闭cURL句柄
curl_close($ch1);
curl_close($ch2);
?>
```

PHP curl_share_init函数

(PHP 5 >= 5.5.0)

curl_share_init — 初始化一个 cURL 共享句柄

说明

```
resource curl_share_init ( void )
```

允许两个 cURL 句柄分享数据。

参数

此函数没有参数。

返回值

返回"cURL共享句柄"的资源。

实例

该实例将创建一个cURL共享句柄，并添加两个 cURL 句柄，两个句柄共享cookie数据。

```
<?php
// 创建cURL共享句柄并设置cookie数据
$sh = curl_share_init();
curl_share_setopt($sh, CURLSHOPT_SHARE, CURL_LOCK_DATA_COOKIE);

// 初始化第一个cURL句柄并指定它为共享句柄
$ch1 = curl_init("http://www.w3cschool.cc/");
curl_setopt($ch1, CURLOPT_SHARE, $sh);

// 执行第一个cURL句柄
curl_exec($ch1);

// 初始化第二个cURL句柄并指定它为共享句柄
$ch2 = curl_init("http://php.net/");
curl_setopt($ch2, CURLOPT_SHARE, $sh);

// 执行第二个cURL句柄
//  所有 $ch1 句柄的数据在 $ch2 句柄中共享
curl_exec($ch2);

// 关闭cURL共享句柄
curl_share_close($sh);

// 关闭cURL句柄
curl_close($ch1);
curl_close($ch2);
?>
```

PHP curl_share_setopt函数

(PHP 5 >= 5.5.0)

curl_share_setopt — 设置 cURL 共享句柄的一个选项。

说明

```
bool curl_share_setopt ( resource $sh , int $option , string $value )
```

设置 cURL 共享句柄的一个选项。

参数

sh

通过 curl_share_init() 初始化的共享句柄。

option

选项	描述
`CURLSHOPT_SHARE`	指定共享的数据类型
`CURLSHOPT_UNSHARE`	指定不共享的数据类型

value

值	描述
`CURL_LOCK_DATA_COOKIE`	共享cookie数据
`CURL_LOCK_DATA_DNS`	共享 DNS 缓存。
`CURL_LOCK_DATA_SSL_SESSION`	共享 SSL session ID, 减少连接到相同的服务器花费在 SSL 握手时的时间。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

该实例将创建一个cURL共享句柄，并添加两个 cURL 句柄，两个句柄共享cookie数据。

```
<?php
// 创建cURL共享句柄并设置cookie数据
$sh = curl_share_init();
curl_share_setopt($sh, CURLSHOPT_SHARE, CURL_LOCK_DATA_COOKIE);

// 初始化第一个cURL句柄并指定它为共享句柄
$ch1 = curl_init("http://www.w3cschool.cc/");
curl_setopt($ch1, CURLOPT_SHARE, $sh);

// 执行第一个cURL句柄
curl_exec($ch1);

// 初始化第二个cURL句柄并指定它为共享句柄
$ch2 = curl_init("http://php.net/");
curl_setopt($ch2, CURLOPT_SHARE, $sh);

// 执行第二个cURL句柄
//  所有 $ch1 句柄的数据在 $ch2 句柄中共享
curl_exec($ch2);

// 关闭cURL共享句柄
curl_share_close($sh);

// 关闭cURL句柄
curl_close($ch1);
curl_close($ch2);
?>
```

PHP curl_strerror函数

(PHP 5 >= 5.5.0)

curl_strerror — 返回错误码的描述。

说明

```
string curl_strerror ( int $errornum )
```

返回错误码的文本描述信息。

参数

errornum

一个 ? cURL 错误码的常量。

返回值

返回错误码描述信息，非法错误码返回NULL 。

实例

```
<?php
// 创建一个拼写错误的URL的cURL句柄
$ch = curl_init("htp://example.com/");

// 发送请求
curl_exec($ch);

// 检查错误代码并显示错误信息
if($errno = curl_errno($ch)) {
    $error_message = curl_strerror($errno);
    echo "cURL error ({$errno}):\n {$error_message}";
}

// 关闭句柄
curl_close($ch);
?>
```

以上例程会输出：

```
cURL error (1):
 Unsupported protocol
```

PHP curl_unescape函数

(PHP 5 >= 5.5.0)

curl_unescape — 解码经过URL编码的字符串。

说明

```
string curl_unescape ( resource $ch , string $str )
```

解码经过URL编码的字符串。

注意： curl_unescape() 不能解码加号 (+) 为空格, urldecode() 可以。

参数

ch

由 curl_init() 返回的 cURL 句柄。

str

URL 编码的字符串

返回值

返回解码字符串或者在失败时返回 FALSE。

实例

```
<?php
// 创建一个curl句柄
$ch = curl_init('http://example.com/redirect.php');

// 发送 HTTP 请求
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
curl_exec($ch);

// 获得最后一个有效的URL
$effective_url = curl_getinfo($ch, CURLINFO_EFFECTIVE_URL);
// ie. "http://example.com/show_location.php?loc=M%C3%BCnchen"

// 解码URL
$effective_url_decoded = curl_unescape($ch, $effective_url);
// "http://example.com/show_location.php?loc=München"

// 关闭句柄
curl_close($ch);
?>
```

PHP curl_version函数

(PHP 5 >= 5.5.0)

curl_version — 获取cURL版本信息。

说明

```
array curl_version ([ int $age = CURLVERSION_NOW ] )
```

返回关于cURL的版本信息。

参数

age

返回值

返回一个相关的数组包含如下元素：

Indice	值描述
version_number	cURL 24位版本号
version	cURL 版本号，字符串形式
ssl_version_number	OpenSSL 24 位版本号
ssl_version	OpenSSL 版本号，字符串形式
libz_version	zlib 版本号，字符串形式
host	关于编译cURL主机的信息
age	
features	一个*CURL_VERSION_XXX*常量的位掩码
protocols	一个cURL支持的协议名字的数组

实例

这个范例将会检查当前cURL版本使用curl_version()返回的'features'位掩码中哪些特性是可用的。

```
<?php
// 获取cURL版本数组
$version = curl_version();

// 在cURL编译版本中使用位域来检查某些特性
$bitfields = Array(
            'CURL_VERSION_IPV6',
            'CURL_VERSION_KERBEROS4',
            'CURL_VERSION_SSL',
            'CURL_VERSION_LIBZ'
            );

foreach($bitfields as $feature)
{
    echo $feature . ($version['features'] & constant($feature) ? ' matches' : ' does not
    echo PHP_EOL;
}
?>
```

PHP Date / Time 函数

PHP Date / Time 简介

date/time 函数允许您提取并格式化服务器上的日期和时间。

注释：这些函数依赖于服务器的本地设置。

安装

date/time 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

Runtime 配置

日期/时间函数的行为受到 php.ini 中设置的影响。

Date/Time 配置选项：

名称	默认	描述	可改变
date.default_latitude	"31.7667"	规定默认纬度（从 PHP 5 开始可用）。date_sunrise() 和 date_sunset() 使用该选项。	PHP_INI_ALL
date.default_longitude	"35.2333"	规定默认经度（从 PHP 5 开始可用）。date_sunrise() 和 date_sunset() 使用该选项。	PHP_INI_ALL
date.sunrise_zenith	"90.83"	规定日出天顶（从 PHP 5 开始可用）。date_sunrise() 和 date_sunset() 使用该选项。	PHP_INI_ALL
date.sunset_zenith	"90.83"	规定日落天顶（从 PHP 5 开始可用）。date_sunrise() 和 date_sunset() 使用该选项。	PHP_INI_ALL
date.timezone	""	规定默认时区（从 PHP 5.1 开始可用）。	PHP_INI_ALL

PHP Date / Time 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
checkdate()	验证格利高里日期。	3
date_default_timezone_get()	返回默认时区。	5
date_default_timezone_set()	设置默认时区。	5
date_sunrise()	返回给定的日期与地点的日出时间。	5
date_sunset()	返回给定的日期与地点的日落时间。	5
date()	格式化本地时间／日期。	3
getdate()	返回日期／时间信息。	3
gettimeofday()	返回当前时间信息。	3
gmdate()	格式化 GMT/UTC 日期/时间。	3
gmmktime()	取得 GMT 日期的 UNIX 时间戳。	3
gmstrftime()	根据本地区域设置格式化 GMT/UTC 时间／日期。	3
idate()	将本地时间/日期格式化为整数	5
localtime()	返回本地时间。	4
microtime()	返回当前时间的微秒数。	3
mktime()	返回一个日期的 Unix 时间戳。	3
strftime()	根据区域设置格式化本地时间／日期。	3
strptime()	解析由 strftime 生成的日期／时间。	5
strtotime()	将任何英文文本的日期或时间描述解析为 Unix 时间戳。	3
time()	返回当前时间的 Unix 时间戳。	3

PHP Date / Time 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
DATE_ATOM	原子钟格式 (如: 2005-08-15T16:13:03+0000)	
DATE_COOKIE	HTTP Cookies 格式 (如: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_ISO8601	ISO-8601 (如: 2005-08-14T16:13:03+0000)	
DATE_RFC822	RFC 822 (如: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_RFC850	RFC 850 (如: Sunday, 14-Aug-05 16:13:03 UTC)	
DATE_RFC1036	RFC 1036 (如: Sunday, 14-Aug-05 16:13:03 UTC)	
DATE_RFC1123	RFC 1123 (如: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_RFC2822	RFC 2822 (如: Sun, 14 Aug 2005 16:13:03 +0000)	
DATE_RSS	RSS (如: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_W3C	World Wide Web Consortium (如: 2005-08-14T16:13:03+0000)	

PHP checkdate() 函数

定义和用法

checkdate() 函数验证一个格里高里日期。

如果指定的值合法，则该函数返回 true，否则返回 false。

日期在下列情况下为合法：

- month 介于且包括 1 - 12
- Day 的值在给定的 month 所应该具有的天数范围之内，闰年已经考虑进去了。
- year 介于且包括 1 到 32767

语法

```
checkdate(month,day,year)
```

参数	描述
month	必需。规定月。
day	必需。规定日。
year	必需。规定年。

例子

```
<?php
var_dump(checkdate(12,31,2000));
var_dump(checkdate(2,29,2003));
var_dump(checkdate(2,29,2004));
?>
```

输出：

```
bool(true)
bool(false)
bool(true)
```

PHP date_default_timezone_get() 函数

定义和用法

date_default_timezone_get() 函数返回脚本中所有日期时间函数所使用的默认时区。

语法

```
date_default_timezone_get(void)
```

参数	描述
void	可选。

说明

本函数返回默认时区，使用如下“假定”的顺序：

- 用 date_default_timezone_set() 函数设定的时区（如果设定了的话）
- TZ 环境变量（如果非空）
- date.timezone 配置选项（如果设定了的话）
- 自己推测（如果操作系统支持）
- 如果以上选择都不成功，则返回 UTC

例子

```
<?php
echo(date_default_timezone_get());
?>
```

输出：

```
Europe/Paris
```

PHP date_default_timezone_set() 函数

定义和用法

date_default_timezone_set() 函数设置用在脚本中所有日期/时间函数的默认时区。

语法

```
date_default_timezone_set(timezone)
```

参数	描述
timezone	必需。时区标识符，比如 "UTC" 或 "Europe/Paris"。合法时区的列表：http://www.php.net/manual/en/timezones.php

说明

注释：自 PHP 5.1.0 起（此版本日期时间函数被重写了），如果时区不合法则每个对日期时间函数的调用都会产生一条 E_NOTICE 级别的错误信息，如果使用系统设定或 TZ 环境变量则还会产生 E_STRICT 级别的信息。

例子

```
<?php
echo(date_default_timezone_set("Europe/Paris"));
?>
```

输出：

```
1
```

PHP date_sunrise() 函数

定义和用法

date_sunrise() 函数返回指定的日期与地点的日出时间。

语法

```
date_sunrise(timestamp,format,latitude,longitude,zenith,gmt_offset)
```

参数	描述
timestamp	必需。
format	可选。规定如何返回结果： SUNFUNCS_RET_STRING (以 string 格式返回结果，比如 16:46) SUNFUNCS_RET_DOUBLE (以 float 格式返回结果，比如 16.78243132) SUNFUNCS_RET_TIMESTAMP (以 integer 格式（时间戳）返回结果，比如 1095034606)
latitude	可选。规定地点的纬度。默认是指北纬。因此如果要指定南纬，必须传递一个负值。
longitude	可选。规定地点的经度。默认是指东经。因此如果要指定西经，必须传递一个负值。
zenith	可选。
gmt_offset	可选。规定 GMT 与本地时间的差值。单位是小时。

例子

```
<?php
//计算葡萄牙里斯本的日出时间
//Latitude: 北纬 38.4 度
//Longitude: 西经 9 度
//Zenith ~= 90
//offset: +1 GMT
echo("Date: " . date("D M d Y") . "<br />");
echo("Sunrise time: ");
echo(date_sunrise(time(),SUNFUNCS_RET_STRING,38.4,-9,90,1));
?>
```

输出：

```
Date: Tue Jan 24 2006
Sunrise time: 08:52
```

PHP date_sunset() 函数

定义和用法

date_sunset() 函数返回指定的日期与地点的日落时间。

语法

```
date_sunset(timestamp,format,latitude,longitude,zenith,gmt_offset)
```

参数	描述
timestamp	必需。
format	可选。规定如何返回结果： SUNFUNCS_RET_STRING (以 string 格式返回结果，比如 16:46) SUNFUNCS_RET_DOUBLE (以 float 格式返回结果，比如 16.78243132) SUNFUNCS_RET_TIMESTAMP (以 integer 格式（时间戳）返回结果，比如 1095034606)
latitude	可选。规定地点的纬度。默认是指北纬。因此如果要指定南纬，必须传递一个负值。
longitude	可选。规定地点的经度。默认是指东经。因此如果要指定西经，必须传递一个负值。
zenith	可选。
gmt_offset	可选。规定 GMT 与本地时间的差值。单位是小时。

例子

```
<?php
//计算葡萄牙里斯本的日出时间
//Latitude: 北纬 38.4 度
//Longitude: 西经 9 度
//Zenith ~= 90
//offset: +1 GMT
echo("Date: " . date("D M d Y") . "<br />");
echo("Sunrise time: ");
echo(date_sunset(time(),SUNFUNCS_RET_STRING,38.4,-9,90,1));
?>
```

输出：

```
Date: Tue Jan 24 2006
Sunrise time: 18:44
```

PHP date() 函数

定义和用法

date() 函数格式化一个本地时间／日期。

语法

```
date(format,timestamp)
```

参数	描述
format	必需。规定如何返回结果。
timestamp	可选。

例子

```
<?php
echo("Result with date():<br />");
echo(date("l") . "<br />");
echo(date("l dS \of F Y h:i:s A") . "<br />");
echo("Oct 3,1975 was on a ".date("l", mktime(0,0,0,10,3,1975))."<br />");
echo(date(DATE_RFC822) . "<br />");
echo(date(DATE_ATOM,mktime(0,0,0,10,3,1975)) . "<br /><br />");

echo("Result with gmdate():<br />");
echo(gmdate("l") . "<br />");
echo(gmdate("l dS \of F Y h:i:s A") . "<br />");
echo("Oct 3,1975 was on a ".gmdate("l", mktime(0,0,0,10,3,1975))."<br />");
echo(gmdate(DATE_RFC822) . "<br />");
echo(gmdate(DATE_ATOM,mktime(0,0,0,10,3,1975)) . "<br />");
?>
```

输出：

```
Result with date():
Tuesday
Tuesday 24th of January 2006 02:41:22 PM
Oct 3,1975 was on a Friday
Tue, 24 Jan 2006 14:41:22 CET
1975-10-03T00:00:00+0100

Result with gmdate():
Tuesday
Tuesday 24th of January 2006 01:41:22 PM
Oct 3,1975 was on a Thursday
Tue, 24 Jan 2006 13:41:22 GMT
1975-10-02T23:00:00+0000
```

PHP getdate() 函数

定义和用法

getdate() 函数取得日期／时间信息。

语法

```
getdate(timestamp)
```

参数	描述
timestamp	可选。规定 Unix 时间格式中的时间。

说明

返回一个根据 *timestamp* 得出的包含有日期信息的结合数组。如果没有给出时间戳，则认为是当前本地时间。

数组中的单元如下：

键名	说明	返回值例子
"seconds"	秒的数字表示	0 到 59
"minutes"	分钟的数字表示	0 到 59
"hours"	小时的数字表示	0 到 23
"mday"	月份中第几天的数字表示	1 到 31
"wday"	星期中第几天的数字表示	0（表示星期天）到 6（表示星期六）
"mon"	月份的数字表示	1 到 12
"year"	4 位数字表示的完整年份	例如：1999 或 2003
"yday"	一年中第几天的数字表示	0 到 365
"weekday"	星期几的完整文本表示	Sunday 到 Saturday
"month"	月份的完整文本表示	January 到 December
0	自从 Unix 纪元开始至今的秒数，和 time() 的返回值以及用于 date() 的值类似。	系统相关，典型值为从 -2147483648 到 2147483647。

例子

例子 1

```
<?php
print_r(getdate());
?>
```

输出：

```
Array
(
[seconds] => 45
[minutes] => 52
[hours] => 14
[mday] => 24
[wday] => 2
[mon] => 1
[year] => 2006
[yday] => 23
[weekday] => Tuesday
[month] => January
[0] => 1138110765
)
```

例子 2

```
<?php
$my_t=getdate(date("U"));
print("$my_t[weekday], $my_t[month] $my_t[mday], $my_t[year]");
?>
```

输出：

```
Wednesday, January 25, 2006
```

PHP gettimeofday() 函数

定义和用法

gettimeofday() 函数返回一个包含当前时间信息的数组。

自 PHP 5.1.0 起有个可选参数 *return_float*，当其设置为 TRUE 时，gettimeofday() 会返回一个浮点数。

所返回的数组键的含义是：

- "sec" - 自 Unix 纪元起的秒数
- "usec" - 微秒数
- "minuteswest" - 格林威治向西的分钟数
- "dsttime" - 夏令时修正的类型

语法

```
gettimeofday(return_float)
```

参数	描述
return_float	可选。当其设置为 TRUE 时，gettimeofday() 会返回一个浮点数。

例子

例子 1

```
<?php
echo(gettimeofday(true) . "<br /><br />");
print_r(gettimeofday());
?>
```

输出：

```
1138111447.4

Array
(
[sec] => 1138111447
[usec] => 395863
[minuteswest] => -60
[dsttime] => 0
)
```

例子 2

```
<?php
$my_t=gettimeofday();
print("$my_t[sec].$my_t[usec]");
?>
```

输出：

```
1138197006.988273
```

PHP gmdate() 函数

定义和用法

gmdate() 函数格式化 GMT/UTC 日期/时间。

同 date() 函数 类似，不同的是返回的时间是格林威治标准时（GMT）。

语法

```
gmdate(format,timestamp)
```

参数	描述
format	可选。规定如何返回结果。
timestamp	可选。

提示和注释

注释：在 PHP 5.1.0 之前，负的时间戳（1970 年之前的日期）在某些系统下（例如 Windows）不能工作。

例子

例子 1

当在中国（GMT +0800）运行以下程序时，第一行显示“Jan 01 2000 00:00:00”，而第二行显示“Dec 31 1999 16:00:00”。

```
<?php
echo date("M d Y H:i:s", mktime (0,0,0,1,1,2000));
echo gmdate("M d Y H:i:s", mktime (0,0,0,1,1,2000));
?>
```

输出：

```
Jan 01 2000 00:00:00
Dec 31 1999 16:00:00
```

例子 2

```
<?php
echo("Result with date():<br />");
echo(date("l") . "<br />");
echo(date("l dS \of F Y h:i:s A") . "<br />");
echo("Oct 3,1975 was on a ".date("l", mktime(0,0,0,10,3,1975))."<br />");
echo(date(DATE_RFC822) . "<br />");
echo(date(DATE_ATOM,mktime(0,0,0,10,3,1975)) . "<br /><br />");

echo("Result with gmdate():<br />");
echo(gmdate("l") . "<br />");
echo(gmdate("l dS \of F Y h:i:s A") . "<br />");
echo("Oct 3,1975 was on a ".gmdate("l", mktime(0,0,0,10,3,1975))."<br />");
echo(gmdate(DATE_RFC822) . "<br />");
echo(gmdate(DATE_ATOM,mktime(0,0,0,10,3,1975)) . "<br />");
?>
```

输出：

```
Result with date():
Tuesday
Tuesday 24th of January 2006 02:41:22 PM
Oct 3,1975 was on a Friday
Tue, 24 Jan 2006 14:41:22 CET
1975-10-03T00:00:00+0100

Result with gmdate():
Tuesday
Tuesday 24th of January 2006 01:41:22 PM
Oct 3,1975 was on a Thursday
Tue, 24 Jan 2006 13:41:22 GMT
1975-10-02T23:00:00+0000
```

PHP gmmktime() 函数

定义和用法

gmmktime() 函数取得 GMT 日期的 UNIX 时间戳。

与 mktime() 类似，不同的是返回值是格林威治标准时的时间戳。

参数总是表示 GMT 日期，因此 *is_dst* 对结果没有影响。

与 mktime() 一样，参数可以从右到左依次空着，空着的参数会被设为相应的当前 GMT 值。

语法

```
gmmktime(hour,minute,second,month,day,year,is_dst)
```

参数	描述
hour	可选。规定小时。
minute	可选。规定分钟。
second	可选。规定秒。
month	可选。规定用数字表示的月。
day	可选。规定天。
year	可选。规定年。在某些系统上，合法值介于 1901 - 2038 之间。不过在 PHP 5 中已经不存在这个限制了。
is_dst	可选。如果时间在日光节约时间(DST)期间，则设置为1，否则设置为0，若未知，则设置为-1。自 5.1.0 起，is_dst 参数被废弃。因此应该使用新的时区处理特性。

提示和注释

注释：gmmktime() 内部使用了 mktime()，因此只有转换成本地时间也合法的时间才能用在其中。

例子

```
<?php
$my_birthday = gmmktime(0,0,0,10,3,1975);
print($my_birthday . "<br />");
print(date("M-d-Y",$my_birthday));
?>
```

输出：

```
181526400
Oct-03-1975
```

PHP gmstrftime() 函数

定义和用法

gmstrftime() 函数根据本地区域设置格式化 GMT/UTC 时间／日期。

语法

```
gmstrftime(format,timestamp)
```

参数	描述
format	可选。规定如何返回结果。
timestamp	可选。

提示和注释

提示：与 strftime() 的行为相同，不同的是返回时间是格林威治标准时（GMT）。

例子

输出 strftime() 和 gmstrftime() 的结果：

```
<?php
echo(strftime("%b %d %Y %X", mktime(20,0,0,12,31,98)));
echo(gmstrftime("%b %d %Y %X", mktime(20,0,0,12,31,98)));

//输出当前日期、时间和时区
echo(gmstrftime("It is %a on %b %d, %Y, %X time zone: %Z",time()));
?>
```

输出：

```
Dec 31 1998 20:00:00
Dec 31 1998 19:00:00
It is Wed on Jan 25, 2006, 11:32:10 time zone: W. Europe Standard Time
```

PHP idate() 函数

定义和用法

idate() 函数将本地时间/日期格式化为整数。

语法

```
strftime(format,timestamp)
```

参数	描述
format	可选。规定如何返回结果。
timestamp	可选。

说明

根据给定的格式字符对 *timestamp* 进行格式化，并返回数字结果。

timestamp 为可选项，默认值为本地当前时间，即 time() 的值。

提示和注释

注释：与 date() 不同，idate() 只接受一个字符作为 *format* 参数。

format 参数可识别以下字符

format 字符	描述
B	Swatch Beat/Internet Time
d	月份中的第几天
h	小时（12 小时格式）
H	小时（24 小时格式）
i	分钟
I	如果启用夏时制则返回 1，否则返回 0
L	如果是闰年则返回 1，否则返回 0
m	月份的数字
s	秒数
t	本月的总天数
U	自 Unix 纪元（January 1 1970 00:00:00 GMT）起的秒数——这和 time() 作用相同
w	星期中的第几天（星期天是 0）
W	ISO-8601 格式年份中的第几个星期，每星期从星期一开始
y	年份（1 或 2 位数字——见下面说明）
Y	年份（4 位数字）
z	年份中的第几天
Z	以秒为单位的时区偏移量

例子

```
<?php
echo(idate("Y"));
?>
```

输出：

```
2008
```

PHP localtime() 函数

定义和用法

localtime() 函数返回本地时间（一个数组）。

localtime() 的第一个参数是时间戳，如果没有给出则使用从 time() 返回的当前时间。

第二个参数是 *is_associative*，如果设为 false 或未提供则返回的是普通的数字索引数组。如果该参数设为 true 则 localtime() 函数返回一个关联数组。

关联数组中不同的键名是：

- "tm_sec" - 秒数
- "tm_min" - 分钟数
- "tm_hour" - 小时
- "tm_mday" - 月份中的第几日
- "tm_mon" - 年份中的第几个月，从 0 开始表示一月
- "tm_year" - 年份，从 1900 开始
- "tm_wday" - 星期中的第几天
- "tm_yday" - 一年中的第几天
- "tm_isdst" - 夏令时当前是否生效

注释：月份从 0（一月）到 11（十二月），星期数从 0（星期天）到 6（星期六）。

语法

```
localtime(timestamp,is_associative)
```

参数	描述
timestamp	可选。规定被格式化的日期或时间。若未规定 timestamp，则使用当前的本地时间。
is_associative	可选。规定返回索引数组还是关联数组。

例子

```
<?php
$localtime = localtime();
$localtime_assoc = localtime(time(), true);
print_r($localtime);
print_r($localtime_assoc);
?>
```

输出：

```
Array
(
    [0] => 24
    [1] => 3
    [2] => 19
    [3] => 3
    [4] => 3
    [5] => 105
    [6] => 0
    [7] => 92
    [9] => 1
)

Array
(
    [tm_sec] => 24
    [tm_min] => 3
    [tm_hour] => 19
    [tm_mday] => 3
    [tm_mon] => 3
    [tm_year] => 105
    [tm_wday] => 0
    [tm_yday] => 92
    [tm_isdst] => 1
)
```

PHP microtime() 函数

定义和用法

microtime() 函数返回当前 Unix 时间戳和微秒数。

语法

```
microtime(get_as_float)
```

参数	描述
get_as_float	如果给出了 get_as_float 参数并且其值等价于 TRUE，该函数将返回一个浮点数。

说明

本函数仅在支持 gettimeofday() 系统调用的操作系统下可用。

如果调用时不带可选参数，本函数以 "msec sec" 的格式返回一个字符串，其中 sec 是自 Unix 纪元（0:00:00 January 1, 1970 GMT）起到现在的秒数，msec 是微秒部分。字符串的两部分都是以秒为单位返回的。

例子

```
<?php
echo(microtime());
?>
```

输出：

```
0.25139300 1138197510
```

PHP mktime() 函数

定义和用法

mktime() 函数返回一个日期的 Unix 时间戳。

参数总是表示 GMT 日期，因此 *is_dst* 对结果没有影响。

参数可以从右到左依次空着，空着的参数会被设为相应的当前 GMT 值。

语法

```
mktime(hour,minute,second,month,day,year,is_dst)
```

参数	描述
hour	可选。规定小时。
minute	可选。规定分钟。
second	可选。规定秒。
month	可选。规定用数字表示的月。
day	可选。规定天。
year	可选。规定年。在某些系统上，合法值介于 1901 - 2038 之间。不过在 PHP 5 中已经不存在这个限制了。
is_dst	可选。如果时间在日光节约时间(DST)期间，则设置为1，否则设置为0，若未知，则设置为-1。自 5.1.0 起，is_dst 参数被废弃。因此应该使用新的时区处理特性。

提示和注释

注释：在 PHP 5.1 之前，如果该函数的参数非法，则会返回 false。

例子

mktime() 函数对于日期运算和验证非常有用。它可以自动校正越界的输入：

```
<?php
echo(date("M-d-Y",mktime(0,0,0,12,36,2001)));
echo(date("M-d-Y",mktime(0,0,0,14,1,2001)));
echo(date("M-d-Y",mktime(0,0,0,1,1,2001)));
echo(date("M-d-Y",mktime(0,0,0,1,1,99)));
?>
```

输出：

```
Jan-05-2002
Feb-01-2002
Jan-01-2001
Jan-01-1999
```

PHP strftime() 函数

定义和用法

strftime() 函数根据区域设置格式化本地时间／日期。

语法

```
strftime(format,timestamp)
```

参数	描述
format	可选。规定如何返回结果。
timestamp	可选。

提示和注释

提示：与 gmstrftime() 的行为相同，不同的是返回时间是本地时间。

例子

输出 strftime() 和 gmstrftime() 的结果：

```
<?php
echo(strftime("%b %d %Y %X", mktime(20,0,0,12,31,98)));
echo(gmstrftime("%b %d %Y %X", mktime(20,0,0,12,31,98)));

//输出当前日期、时间和时区
echo(gmstrftime("It is %a on %b %d, %Y, %X time zone: %Z",time()));
?>
```

输出：

```
Dec 31 1998 20:00:00
Dec 31 1998 19:00:00
It is Wed on Jan 25, 2006, 11:32:10 time zone: W. Europe Standard Time
```

PHP strptime() 函数

定义和用法

strptime() 函数解析由 strftime() 生成的日期／时间。

语法

```
strptime(date,format)
```

参数	描述
date	要解析的字符串（例如从 strftime() 返回的）。
format	date 所使用的格式（与 strftime() 中所使用的相同）。

说明

strptime() 返回一个将 *date* 解析后的数组，如果出错返回 FALSE。

月份和星期几的名字以及其它与语种有关的字符串对应于 setlocale()设定的当前区域（LC_TIME）。

数组中包含以下单元：

键名	说明
tm_sec	当前分钟内的秒数（0-61）
tm_min	当前小时内的分钟数（0-59）
tm_hour	午夜起的小时数（0-23）
tm_mday	月份中的第几天（1-31）
tm_mon	自一月起过了几个月（0-11）
tm_year	自 1900 年起过了几年
tm_wday	自星期天起过了几天（0-6）
tm_yday	本年自一月一日起过了多少天（0-365）
unparsed date	中未能通过指定的 format 识别的部分

例子

输出 strftime() 和 strptime() 的结果：

```
<?php
$format="%d/%m/%Y %H:%M:%S";
$strf=strftime($format);
echo("$strf");
print_r(strptime($strf,$format));
?>
```

输出：

```
03/10/2005 13:23:44
Array
(
[tm_sec] => 44
[tm_min] => 23
[tm_hour] => 13
[tm_mday] => 3
[tm_mon] => 9
[tm_year] => 105
[tm_wday] => 0
[tm_yday] => 276
[unparsed] =>
)
```

PHP strtotime() 函数

定义和用法

strtotime() 函数将任何英文文本的日期时间描述解析为 Unix 时间戳。

语法

```
strtotime(time,now)
```

参数	描述
time	规定要解析的时间字符串。
now	用来计算返回值的时间戳。如果省略该参数，则使用当前时间。

说明

该函数预期接受一个包含美国英语日期格式的字符串并尝试将其解析为 Unix 时间戳（自 January 1 1970 00:00:00 GMT 起的秒数），其值相对于 *now* 参数给出的时间，如果没有提供此参数，则用系统当前时间。

该函数将使用 *TZ* 环境变量（如果有的话）来计算时间戳。自 PHP 5.1.0 起有更容易的方法来定义时区用于所有的日期／时间函数。此过程在 date_default_timezone_get() 函数页面中有说明。

返回值

成功则返回时间戳，否则返回 FALSE。在 PHP 5.1.0 之前本函数在失败时返回 -1。

例子

```
<?php
echo(strtotime("now"));
echo(strtotime("3 October 2005"));
echo(strtotime("+5 hours"));
echo(strtotime("+1 week"));
echo(strtotime("+1 week 3 days 7 hours 5 seconds"));
echo(strtotime("next Monday"));
echo(strtotime("last Sunday"));
?>
```

输出：

```
1138614504
1128290400
1138632504
1139219304
1139503709
1139180400
1138489200
```

PHP time() 函数

定义和用法

time() 函数返回当前时间的 Unix 时间戳。

语法

```
time(void)
```

参数	描述
void	可选。

说明

返回自从 Unix 纪元（格林威治时间 1970 年 1 月 1 日 00:00:00）到当前时间的秒数。

提示和注释

提示：自 PHP 5.1 起在 *$_SERVER['REQUEST_TIME']* 中保存了发起该请求时刻的时间戳。

例子

例子 1

```
<?php
$t=time();
echo($t . "<br />");
echo(date("D F d Y",$t));
?>
```

输出：

```
1138618081
Mon January 30 2006
```

例子 2

```
<?php
$nextWeek = time() + (7 * 24 * 60 * 60); // 7 days; 24 hours; 60 mins; 60secs
echo 'Now:       '. date('Y-m-d') ."\n";
echo 'Next Week: '. date('Y-m-d', $nextWeek) ."\n";
?>
```

输出：

```
Now:       2005-03-30
Next Week: 2005-04-07
```

PHP Directory 函数

PHP Directory 简介

Directory 函数允许您获得关于目录及其内容的信息。

安装

Directory 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP Directory 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
chdir()	改变当前的目录。	3
chroot()	改变当前进程的根目录。	4
dir()	打开一个目录句柄，并返回一个对象。	3
closedir()	关闭目录句柄。	3
getcwd()	返回当前目录。	4
opendir()	打开目录句柄。	3
readdir()	返回目录句柄中的条目。	3
rewinddir()	重置目录句柄。	3
scandir()	列出指定路径中的文件和目录。	5

PHP Directory 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
DIRECTORY_SEPARATOR	3	
PATH_SEPARATOR	4	

PHP chdir() 函数

定义和用法

chdir() 函数把当前的目录改变为指定的目录。

若成功，则该函数返回 true，否则返回 false。

语法

```
chdir(directory)
```

参数	描述
directory	必需。规定新的当前目录。

例子

```
<?php
//获得当前目录
echo getcwd();
echo "<br />";

//改变为 images 目录
chdir("images");
echo "<br />";
echo getcwd();
?>
```

输出：

```
C:\testweb\main
C:\testweb\main\images
```

PHP chroot() 函数

定义和用法

chroot() 函数把当前进程的根目录改变为指定的目录。

若成功，则该函数返回 true，否则返回 false。

语法

```
chroot(directory)
```

参数	描述
directory	必需。规定新的根目录。

提示和注释

提示：该函数没有在 Windows 平台上实现。

PHP dir() 函数

定义和用法

dir() 函数打开一个目录句柄，并返回一个对象。这个对象包含三个方法：read() , rewind() 以及 close()。

若成功，则该函数返回一个目录流，否则返回 false 以及一个 error。可以通过在函数名前加上 "@" 来隐藏 error 的输出。

语法

```
dir(directory)
```

参数	描述
directory	必需。规定目录。

例子

例子 1

```
<?php
//打开 images 目录
$dir = dir("images");

//列出 images 目录中的文件
while (($file = $dir->read()) !== false)
{
echo "filename: " . $file . "<br />";
}

$dir->close();
?>
```

输出：

```
filename: .
filename: ..
filename: cat.gif
filename: dog.gif
filename: food
filename: horse.gif
```

例子 2

如果 dir() 的输出失败，本例会隐藏错误：

```
<?php
//打开 images 目录
$dir = @ dir("images");

//列出 images 目录中的文件
while (($file = $dir->read()) !== false)
  {
  echo "filename: " . $file . "<br />";
  }

$dir->close();
?>
```

输出：

```
filename: .
filename: ..
filename: cat.gif
filename: dog.gif
filename: food
filename: horse.gif
```

PHP closedir() 函数

定义和用法

closedir() 函数关闭由 opendir() 函数打开的目录句柄。

语法

```
closedir(dir_stream)
```

参数	描述
dir_stream	必需。规定要关闭的目录句柄。

例子

```
<?php
//打开 images 目录
$dir = opendir("images");

//列出 images 目录中的文件
while (($file = readdir($dir)) !== false)
  {
  echo "filename: " . $file . "<br />";
  }
  closedir($dir);
?>
```

输出：

```
filename: .
filename: ..
filename: cat.gif
filename: dog.gif
filename: food
filename: horse.gif
```

PHP getcwd() 函数

定义和用法

getcwd() 函数返回当前目录。

若成功，则返回当前工作目录，否则返回 false。

语法

```
getcwd()
```

例子

```
<?php
echo getcwd();
?>
```

输出：

```
C:\testweb\main
```

PHP opendir() 函数

定义和用法

opendir() 函数打开一个目录句柄，可由 closedir()，readdir() 和 rewinddir() 使用。

若成功，则该函数返回一个目录流，否则返回 false 以及一个 error。可以通过在函数名前加上 "@" 来隐藏 error 的输出。

语法

```
opendir(path,context)
```

参数	描述
path	必需。规定要打开的目录路径。
context	可选。规定目录句柄的环境。context 是可修改目录流的行为的一套选项。

提示和注释

注释：从 PHP 5.0.0 开始，*path* 参数支持 ftp:// URL wrapper。

注释：在 PHP 4.3.0 中， *path* 参数可以是任何支持目录列表的 URL，不过在 PHP 4 中只有 file:// URL wrapper 支持此功能。

例子

例子 1

```
<?php
//打开 images 目录
$dir = opendir("images");

//列出 images 目录中的文件
while (($file = readdir($dir)) !== false)
  {
  echo "filename: " . $file . "<br />";
  }
  closedir($dir);
?>
```

输出：

```
filename: .
filename: ..
filename: cat.gif
filename: dog.gif
filename: food
filename: horse.gif
```

例子 2

如果 opendir() 的输出失败，本例会隐藏错误：

```
<?php
//打开 images 目录
$dir = @ opendir("images");

//列出 images 目录中的文件
while (($file = readdir($dir)) !== false)
  {
  echo "filename: " . $file . "<br />";
  }
  closedir($dir);
?>
```

输出：

```
filename: .
filename: ..
filename: cat.gif
filename: dog.gif
filename: food
filename: horse.gif
```

PHP readdir() 函数

定义和用法

readdir() 函数返回由 opendir() 打开的目录句柄中的条目。

若成功，则该函数返回一个文件名，否则返回 false。

语法

```
readdir(dir_stream)
```

参数	描述
dir_stream	必需。规定要使用的目录句柄。

说明

返回目录中下一个文件的文件名。文件名以在文件系统中的排序返回。

例子

```
<?php
//打开 images 目录
$dir = opendir("images");

//列出 images 目录中的文件
while (($file = readdir($dir)) !== false)
  {
  echo "filename: " . $file . "<br />";
  }
  closedir($dir);
?>
```

输出：

```
filename: .
filename: ..
filename: cat.gif
filename: dog.gif
filename: food
filename: horse.gif
```

PHP rewinddir() 函数

定义和用法

rewinddir() 函数重置由 opendir() 打开的目录句柄。

本函数什么都不会返回。

语法

```
rewinddir(dir_stream)
```

参数	描述
dir_stream	必需。规定要使用的目录句柄。

提示和注释

提示：本函数重读目录，并可用于检查目录中的变化。

例子

```
<?php
//打开 images 目录
$dir = opendir("images");

//列出 images 目录中的文件
while (($file = readdir($dir)) !== false)
  {
  echo "filename: " . $file . "<br />";
  }

//重置目录流
rewinddir($dir);

//供检查变化的代码

closedir($dir);
?>
```

PHP scandir() 函数

定义和用法

scandir() 函数返回一个数组，其中包含指定路径中的文件和目录。

若成功，则返回一个数组，若失败，则返回 false。如果 *directory* 不是目录，则返回布尔值 false 并生成一条 E_WARNING 级的错误。

语法

```
scandir(directory,sort,context)
```

参数	描述
directory	必需。规定要扫描的目录。
sort	可选。规定排列顺序。默认是 0 （升序）。如果是 1，则为降序。
context	可选。规定目录句柄的环境。context 是可修改目录流的行为的一套选项。

例子

```
<?php
print_r(scandir("images"));
?>
```

输出：

```
Array
(
[0] => .
[1] => ..
[2] => dog.jpg
[3] => house.jpg
[4] => logo.gif
)
```

PHP Error 和 Logging 函数

PHP Error 和 Logging 简介

error 和 logging 函数允许你对错误进行处理和记录。

error 函数允许用户定义错误处理规则，并修改记录错误的方式。

logging 函数允许用户对应用程序进行日志记录，并把日志消息发送到电子邮件、系统日志或其他的机器。

安装

error 和 logging 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP Error 和 Logging 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
debug_backtrace()	生成 backtrace。	4
debug_print_backtrace()	输出 backtrace。	5
error_get_last()	获得最后发生的错误。	5
error_log()	向服务器错误记录、文件或远程目标发送一个错误。	4
error_reporting()	规定报告哪个错误。	4
restore_error_handler()	恢复之前的错误处理程序。	4
restore_exception_handler()	恢复之前的异常处理程序。	5
set_error_handler()	设置用户自定义的错误处理函数。	4
set_exception_handler()	设置用户自定义的异常处理函数。	5
trigger_error()	创建用户自定义的错误消息。	4
user_error()	trigger_error() 的别名。	4

PHP Error 和 Logging 常量

PHP：指示支持该常量的最早的 PHP 版本。

值	常量	描述	PHP
1	E_ERROR	致命的运行时错误。错误无法恢复。脚本的执行被中断。	
2	E_WARNING	非致命的运行时错误。脚本的执行不会中断。	
4	E_PARSE	编译时语法解析错误。解析错误只应该由解析器生成。	
8	E_NOTICE	运行时提示。可能是错误，也可能在正常运行脚本时发生。	
16	E_CORE_ERROR	由 PHP 内部生成的错误。	4
32	E_CORE_WARNING	由 PHP 内部生成的警告。	4
64	E_COMPILE_ERROR	由 Zend 脚本引擎内部生成的错误。	4
128	E_COMPILE_WARNING	由 Zend 脚本引擎内部生成的警告。	4
256	E_USER_ERROR	由于调用 trigger_error() 函数生成的运行时错误。	4
512	E_USER_WARNING	由于调用 trigger_error() 函数生成的运行时警告。	4
1024	E_USER_NOTICE	由于调用 trigger_error() 函数生成的运行时提示。	4
2048	E_STRICT	运行时提示。对增强代码的互用性和兼容性有益。	5
4096	E_RECOVERABLE_ERROR	可捕获的致命错误。（参阅 set_error_handler()）	5
8191	E_ALL	所有的错误和警告，除了 E_STRICT。	5

PHP debug_backtrace() 函数

定义和用法

PHP debug_backtrace() 函数生成一个 backtrace。

该函数返回一个关联数组。下面是可能返回的元素：

名称	类型	描述
function	字符串	当前的函数名。
line	整数	当前的行号。
file	字符串	当前的文件名。
class	字符串	当前的类名
object	对象	当前对象。
type	字符串	当前的调用类型，可能的调用： 返回: " `->` " - 方法调用 返回: " `::` " - 静态方法调用 返回 `nothing` - 函数调用
args	数组	如果在函数中，列出函数参数。如果在被引用的文件中，列出被引用的文件名。

语法

```
debug_backtrace()
```

例子

```
<?php
function one($str1, $str2)
 {
 two("Glenn", "Quagmire");
 }
function two($str1, $str2)
 {
 three("Cleveland", "Brown");
 }
function three($str1, $str2)
 {
 print_r(debug_backtrace());
 }

one("Peter", "Griffin");
?>
```

输出：

```
Array
(
[0] => Array
 (
 [file] => C:\webfolder\test.php
 [line] => 7
 [function] => three
 [args] => Array
  (
  [0] => Cleveland
  [1] => Brown
  )
 )
[1] => Array
 (
 [file] => C:\webfolder\test.php
 [line] => 3
 [function] => two
 [args] => Array
  (
  [0] => Glenn
  [1] => Quagmire
  )
 )
[2] => Array
 (
 [file] => C:\webfolder\test.php
 [line] => 14
 [function] => one
 [args] => Array
  (
  [0] => Peter
  [1] => Griffin
  )
 )
)
```

PHP debug_print_backtrace() 函数

定义和用法

debug_print_backtrace() 函数输出 backtrace。

语法

```
debug_print_backtrace()
```

例子

```
<?php
function one($str1, $str2)
 {
 two("Glenn", "Quagmire");
 }
function two($str1, $str2)
 {
 three("Cleveland", "Brown");
 }
function three($str1, $str2)
 {
 debug_print_backtrace();
 }

one("Peter", "Griffin");
?>
```

输出：

```
#0 three(Cleveland, Brown) called at [C:\webfolder\test.php:8]
#1 two(Glenn, Quagmire) called at [C:\webfolder\test.php:4]
#2 one(Peter, Griffin) called at [C:\webfolder\test.php:15]
```

PHP error_get_last() 函数

定义和用法

error_get_last() 函数获取最后发生的错误。

该函数以数组的形式返回最后发生的错误。

返回的数组包含 4 个键和值：

- [type] - 错误类型
- [message] - 错误消息
- [file] - 发生错误所在的文件
- [line] - 发生错误所在的行

语法

```
error_get_last()
```

例子

```
<?php
echo $test;
print_r(error_get_last());
?>
```

输出：

```
Array
(
[type] => 8
[message] => Undefined variable: test
[file] => C:\webfolder\test.php
[line] => 2
)
```

PHP error_log() 函数

定义和用法

error_log() 函数向服务器错误记录、文件或远程目标发送一个错误。

若成功，返回 true，否则返回 false。

语法

```
error_log(error,type,destination,headers)
```

参数	描述
error	必需。要记录的错误消息。
type	可选。规定错误记录的类型。可能的记录类型： `0` - 默认。根据在 php.ini 文件中的 error_log 配置，错误被发送到服务器日志系统或文件。 `1` - 错误被发送到 destination 参数中的地址。只有该类型使用 headers 参数。 `2` - 通过 PHP debugging 连接来发送错误。该选项只在 PHP 3 中可用。 `3` - 错误发送到文件目标字符串。
destination	可选。规定向何处发送错误消息。该参数的值依赖于 "type" 参数的值。
headers	可选。只在 "type" 为 1 时使用。规定附加的头部，比如 From, Cc 以及 Bcc。由 CRLF (\r\n) 分隔。注释：在发送电子邮件时，必须包含 From 头部。可以在 php.ini 文件中或者通过此参数设置。

例子

本例发送一封带有自定义错误的电子邮件：

```
<?php
$test=2;

if ($test>1)
{
error_log("A custom error has been triggered",
1,"someone@example.com","From: webmaster@example.com");
}
?>
```

输出：

```
A custom error has been triggered
```

PHP error_reporting() 函数

定义和用法

error_reporting() 设置 PHP 的报错级别并返回当前级别。

语法

```
error_reporting(report_level)
```

如果参数 level 未指定，当前报错级别将被返回。下面几项是 level 可能的值：

值	常量	描述
1	E_ERROR	Fatal run-time errors. Errors that can not be recovered from. Execution of the script is halted
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
4	E_PARSE	Compile-time parse errors. Parse errors should only be generated by the parser
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
16	E_CORE_ERROR	Fatal errors at PHP startup. This is like an E_ERROR in the PHP core
32	E_CORE_WARNING	Non-fatal errors at PHP startup. This is like an E_WARNING in the PHP core
64	E_COMPILE_ERROR	Fatal compile-time errors. This is like an E_ERROR generated by the Zend Scripting Engine
128	E_COMPILE_WARNING	Non-fatal compile-time errors. This is like an E_WARNING generated by the Zend Scripting Engine
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
2048	E_STRICT	Run-time notices. PHP suggest changes to your code to help interoperability and compatibility of the code
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings, except level E_STRICT (E_STRICT will be part of E_ALL as of PHP 6.0)

例子

任意数目的以上选项都可以用“或”来连接（用 OR 或 |），这样可以报告所有需要的各级别错误。例如，下面的代码关闭了用户自定义的错误和警告，执行了某些操作，然后恢复到原始的报错级别：

```
<?php
//禁用错误报告
error_reporting(0);

//报告运行时错误
error_reporting(E_ERROR | E_WARNING | E_PARSE);

//报告所有错误
error_reporting(E_ALL);
?>
```

PHP restore_error_handler() 函数

定义和用法

restore_error_handler() 函数恢复之前的错误处理程序，该程序是由 set_error_handler() 函数改变的。

该函数永远返回 true。

语法

```
restore_error_handler()
```

提示和注释

提示：之前的错误处理程序可能是在错误处理程序或用户自定义函数中构建的。

例子

```
<?php
//custom error handler function
function customError($errno, $errstr, $errfile, $errline)
 {
 echo "<b>Custom error:</b> [$errno] $errstr<br />";
 echo " Error on line $errline in $errfile<br />";
 }

//set user-defined error handler
set_error_handler("customError");

$test=2;

//trigger error
if ($test>1)
 {
 trigger_error("A custom error has been triggered");
 }

//restore built-in error handler
restore_error_handler

//trigger error again
if ($test>1)
 {
 trigger_error("A custom error has been triggered");
 }
?>
```

输出：

```
**Custom error:** [1024] A custom error has been triggered
Error on line 14 in C:\webfolder\test.php

**Notice**: A custom error has been triggered in **
C:\webfolder\test.php** on line **21**
```

PHP restore_exception_handler() 函数

定义和用法

restore_exception_handler() 函数恢复之前的异常处理程序，该程序是由 set_exception_handler() 函数改变的。

该函数永远返回 true。

语法

```
restore_exception_handler()
```

提示和注释

提示：之前的异常处理程序可能是在异常处理程序或用户自定义函数中构建的。

例子

```
<?php
restore_exception_handler();

throw new Exception('Uncaught Exception occured');
?>
```

输出：

```
**Fatal error**: Uncaught exception 'Exception' with message
'Uncaught Exception occured' in C:\webfolder\test.php:4
Stack trace: #0 {main} thrown in **C:\webfolder\test.php** on line **4**
```

PHP set_error_handler() 函数

定义和用法

set_error_handler() 函数设置用户自定义的错误处理函数。

该函数用于创建运行时期间的用户自己的错误处理方法。

该函数会返回旧的错误处理程序，若失败，则返回 null。

语法

```
set_error_handler(error_function,error_types)
```

参数	描述
error_function	必需。规定发生错误时运行的函数。
error_types	可选。规定在哪个错误报告级别会显示用户定义的错误。默认是 "E_ALL"。

提示和注释

提示：如果使用了该函数，会完全绕过标准的 PHP 错误处理函数，如果必要，用户定义的错误处理程序必须终止 (die()) 脚本。

注释：如果在脚本执行前发生错误，由于在那时自定义程序还没有注册，因此就不会用到这个自定义错误处理程序。

例子

```
<?php
//error handler function
function customError($errno, $errstr, $errfile, $errline)
 {
 echo "<b>Custom error:</b> [$errno] $errstr<br />";
 echo " Error on line $errline in $errfile<br />";
 echo "Ending Script";
 die();
 }

//set error handler
set_error_handler("customError");

$test=2;

//trigger error
if ($test>1)
 {
 trigger_error("A custom error has been triggered");
 }
?>
```

输出：

```
**Custom error:** [1024] A custom error has been triggered
Error on line 19 in C:\webfolder\test.php
Ending Script
```

PHP set_exception_handler() 函数

定义和用法

set_exception_handler() 函数设置用户自定义的异常处理函数。

该函数用于创建运行时期间的用户自己的异常处理方法。

该函数会返回旧的异常处理程序，若失败，则返回 null。

语法

```
set_exception_handler(exception_function)
```

参数	描述
error_function	必需。规定未捕获的异常发生时调用的函数。该函数必须在调用 set_exception_handler() 函数之前定义。这个异常处理函数需要需要一个参数，即抛出的 exception 对象。

提示和注释

提示：在这个异常处理程序被调用后，脚本会停止执行。

例子

```
<?php
function myException($exception)
{
echo "<b>Exception:</b> " , $exception->getMessage();
}

set_exception_handler('myException');

throw new Exception('Uncaught Exception occurred');
?>
```

输出：

```
**Exception:** Uncaught Exception occurred
```

PHP trigger_error() 函数

定义和用法

trigger_error() 函数创建用户定义的错误消息。

trigger_error() 用于在用户指定的条件下触发一个错误消息。它与内建的错误处理器一同使用，也可以与由 set_error_handler() 函数创建的用户自定义函数使用。

如果指定了一个不合法的错误类型，该函数返回 false，否则返回 true。

语法

```
trigger_error(error_message,error_types)
```

参数	描述
error_message	必需。规定错误消息。长度限制为 1024 个字符。
error_types	可选。规定错误消息的错误类型。 可能的值： E_USER_ERROR E_USER_WARNING E_USER_NOTICE

例子

```
<?php
$test=2;
if ($test>1)
{
trigger_error("A custom error has been triggered");
}
?>
```

输出：

```
**Notice**: A custom error has been triggered
in **C:\webfolder\test.php** on line **6**
```

PHP Filesystem 函数

PHP Filesystem 简介

Filesystem 函数允许您访问和操作文件系统。

安装

Filesystem 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

Runtime 配置

文件系统函数的行为受到 php.ini 中设置的影响。

文件系统配置选项：

名称	默认	描述	可改变
allow_url_fopen	"1"	本选项激活了 URL 形式的 fopen 封装协议使得可以访问 URL 对象例如文件。默认的封装协议提供用 ftp 和 http 协议来访问远程文件，一些扩展库例如 zlib 可能会注册更多的封装协议。（PHP 4.0.4 版以后可用。）	PHP_INI_SYSTEM
user_agent	NULL	定义 PHP 发送的 User-Agent。（PHP 4.3.0 版以后可用。）	PHP_INI_ALL
default_socket_timeout	"60"	基于 socket 的流的默认超时时间(秒)。（PHP 4.3.0 版以后可用。）	PHP_INI_ALL
from	""	定义匿名 ftp 的密码（您的 email 地址）。	PHP_INI_ALL
auto_detect_line_endings	"0"	当设为 On 时，PHP 将检查通过 fgets() 和 file() 取得的数据中的行结束符号是符合 Unix，MS-DOS，还是 Macintosh 的习惯。这使得 PHP 可以和 Macintosh 系统交互操作，但是默认值是 Off，因为在检测第一行的 EOL 习惯时会有很小的性能损失，而且在 Unix 系统下使用回车符号作为项目分隔符的人们会遭遇向下不兼容的行为。（PHP 4.3.0 版以后可用。）	PHP_INI_ALL

Unix / Windows 兼容性

当在 Unix 平台上规定路径时，正斜杠 (/) 用作目录分隔符。而在 Windows 平台上，正斜杠 (/) 和反斜杠 () 均可使用。

PHP Filesystem 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
basename()	返回路径中的文件名部分	3

basename()	返回路径中的文件名部分。	3
chgrp()	改变文件组。	3
chmod()	改变文件模式。	3
chown()	改变文件所有者。	3
clearstatcache()	清除文件状态缓存。	3
copy()	复制文件。	3
delete()	参见 unlink() 或 unset()。	
dirname()	返回路径中的目录名称部分。	3
disk_free_space()	返回目录的可用空间。	4
disk_total_space()	返回一个目录的磁盘总容量。	4
diskfreespace()	disk_free_space() 的别名。	3
fclose()	关闭打开的文件。	3
feof()	测试文件指针是否到了文件结束的位置。	3
fflush()	向打开的文件输出缓冲内容。	4
fgetc()	从打开的文件中返回字符。	3
fgetcsv()	从打开的文件中解析一行，校验 CSV 字段。	3
fgets()	从打开的文件中返回一行。	3
fgetss()	从打开的文件中读取一行并过滤掉 HTML 和 PHP 标记。	3
file()	把文件读入一个数组中。	3
file_exists()	检查文件或目录是否存在。	3
file_get_contents()	将文件读入字符串。	4
file_put_contents()	将字符串写入文件。	5
fileatime()	返回文件的上次访问时间。	3
filectime()	返回文件的上次改变时间。	3
filegroup()	返回文件的组 ID。	3
fileinode()	返回文件的 inode 编号。	3
filemtime()	返回文件的上次修改时间。	3
fileowner()	文件的 user ID （所有者）。	3
fileperms()	返回文件的权限。	3
filesize()	返回文件大小。	3
filetype()	返回文件类型。	3

flock()	锁定或释放文件。	3
fnmatch()	根据指定的模式来匹配文件名或字符串。	4
fopen()	打开一个文件或 URL。	3
fpassthru()	从打开的文件中读数据，直到 EOF，并向输出缓冲写结果。	3
fputcsv()	将行格式化为 CSV 并写入一个打开的文件中。	5
fputs()	fwrite() 的别名。	3
fread()	读取打开的文件。	3
fscanf()	根据指定的格式对输入进行解析。	4
fseek()	在打开的文件中定位。	3
fstat()	返回关于一个打开的文件的信息。	4
ftell()	返回文件指针的读/写位置	3
ftruncate()	将文件截断到指定的长度。	4
fwrite()	写入文件。	3
glob()	返回一个包含匹配指定模式的文件名/目录的数组。	4
is_dir()	判断指定的文件名是否是一个目录。	3
is_executable()	判断文件是否可执行。	3
is_file()	判断指定文件是否为常规的文件。	3
is_link()	判断指定的文件是否是连接。	3
is_readable()	判断文件是否可读。	3
is_uploaded_file()	判断文件是否是通过 HTTP POST 上传的。	3
is_writable()	判断文件是否可写。	4
is_writeable()	is_writable() 的别名。	3
link()	创建一个硬连接。	3
linkinfo()	返回有关一个硬连接的信息。	3
lstat()	返回关于文件或符号连接的信息。	3
mkdir()	创建目录。	3
move_uploaded_file()	将上传的文件移动到新位置。	4
parse_ini_file()	解析一个配置文件。	4
pathinfo()	返回关于文件路径的信息。	4
pclose()	关闭有 popen() 打开的进程。	3
popen()	打开一个进程。	3

readfile()	读取一个文件，并输出到输出缓冲。	3
readlink()	返回符号连接的目标。	3
realpath()	返回绝对路径名。	4
rename()	重名名文件或目录。	3
rewind()	倒回文件指针的位置。	3
rmdir()	删除空的目录。	3
set_file_buffer()	设置已打开文件的缓冲大小。	3
stat()	返回关于文件的信息。	3
symlink()	创建符号连接。	3
tempnam()	创建唯一的临时文件。	3
tmpfile()	建立临时文件。	3
touch()	设置文件的访问和修改时间。	3
umask()	改变文件的文件权限。	3
unlink()	删除文件。	3

PHP Filesystem 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	**PHP**
GLOB_BRACE		
GLOB_ONLYDIR		
GLOB_MARK		
GLOB_NOSORT		
GLOB_NOCHECK		
GLOB_NOESCAPE		
PATHINFO_DIRNAME		
PATHINFO_BASENAME		
PATHINFO_EXTENSION		
FILE_USE_INCLUDE_PATH		
FILE_APPEND		
FILE_IGNORE_NEW_LINES		
FILE_SKIP_EMPTY_LINES		

PHP basename() 函数

定义和用法

basename() 函数返回路径中的文件名部分。

语法

```
basename(path,suffix)
```

参数	描述
path	必需。规定要检查的路径。
suffix	可选。规定文件扩展名。如果文件有 suffix，则不会输出这个扩展名。

例子

```
<?php
$path = "/testweb/home.php";

//显示带有文件扩展名的文件名
echo basename($path);

//显示不带有文件扩展名的文件名
echo basename($path,".php");
?>
```

输出：

```
home.php
home
```

PHP chgrp() 函数

定义和用法

chgrp() 函数改变文件所属的组。

如果成功则返回 TRUE，否则返回 FALSE。

语法

```
chgrp(file,group)
```

参数	描述
file	必需。规定要检查的文件。
group	可选。规定新的组。可以是组名或组的 ID。

说明

尝试将文件 *file* 所属的组改成 *group* （通过组名或组 ID 指定）。

只有超级用户可以任意修改文件的组，其它用户可能只能将文件的组改成该用户自己所在的组。

例子

```
<?php
chgrp("test.txt","admin")
?>
```

PHP chmod() 函数

定义和用法

chmod() 函数改变文件模式。

如果成功则返回 TRUE，否则返回 FALSE。

语法

```
chmod(file,mode)
```

参数	描述
file	必需。规定要检查的文件。
mode	可选。规定新的权限。mode 参数由 4 个数字组成： 第一个数字永远是 0 第二个数字规定所有者的权限 第二个数字规定所有者所属的用户组的权限 第四个数字规定其他所有人的权限 可能的值（如需设置多个权限，请对下面的数字进行总计）： 1 - 执行权限 2 - 写权限 4 - 读权限

例子

```
<?php
// 所有者可读写，其他人没有任何权限
chmod("test.txt",0600);

// 所有者可读写，其他人可读
chmod("test.txt",0644);

// 所有者有所有权限，其他所有人可读和执行
chmod("test.txt",0755);

// 所有者有所有权限，所有者所在的组可读
chmod("test.txt",0740);
?>
```

PHP chown() 函数

定义和用法

chown() 函数改变指定文件的所有者。

如果成功则返回 TRUE，否则返回 FALSE。

语法

```
chown(file,owner)
```

参数	描述
file	必需。规定要检查的文件。
owner	规定新的所有者。可以是用户名或用户的 ID。

说明

尝试将文件 *file* 的所有者改成用户 *owner*（由用户名或用户 ID 指定）。只有超级用户可以改变文件的所有者。

例子

```
<?php
chown("test.txt","charles")
?>
```

PHP clearstatcache() 函数

定义和用法

clearstatcache() 函数清除文件状态缓存。

clearstatcache() 函数会缓存某些函数的返回信息，以便提供更高的性能。但是有时候，比如在一个脚本中多次检查同一个文件，而该文件在此脚本执行期间有被删除或修改的危险时，你需要清除文件状态缓存，以便获得正确的结果。要做到这一点，就需要使用 clearstatcache() 函数。

会进行缓存的函数，即受 clearstatcache() 函数影响的函数：

- stat()
- lstat()
- file_exists()
- is_writable()
- is_readable()
- is_executable()
- is_file()
- is_dir()
- is_link()
- filectime()
- fileatime()
- filemtime()
- fileinode()
- filegroup()
- fileowner()
- filesize()
- filetype()
- fileperms()

语法

```
clearstatcache()
```

例子

```
<?php
//检查文件大小
echo filesize("test.txt");

$file = fopen("test.txt", "a+");

// 截取文件
ftruncate($file,100);
fclose($file);

//清除缓存并再次检查文件大小
clearstatcache();
echo filesize("test.txt");
?>
```

输出：

```
792
100
```

PHP copy() 函数

定义和用法

copy() 函数拷贝文件。

语法

```
copy(_source_,_destination_)
```

参数	描述
source	必需。规定要复制的文件。
destination	必需。规定复制文件的目的地。

说明

将文件从 *source* 拷贝到 *destination*。如果成功则返回 TRUE，否则返回 FALSE。

提示和注释

提示：如果要移动文件的话，请使用 rename() 函数。

注释：从 PHP 4.3.0 开始，如果启用了 "fopen wrappers" 的话，source 和 *destination* 都可以是 URL。更多信息见 fopen()。如果 *destination* 是一个 URL，则如果封装协议不支持覆盖已有的文件时拷贝操作会失败。

重要事项：如果目标文件已存在，将会被覆盖。

例子

```
<?php
echo copy("source.txt","target.txt");
?>
```

输出：

```
1
```

PHP dirname() 函数

定义和用法

dirname() 函数返回路径中的目录部分。

语法

```
dirname(path)
```

参数	描述
path	必需。规定要检查的路径。

说明

path 参数是一个包含有指向一个文件的全路径的字符串。该函数返回去掉文件名后的目录名。

例子

```
<?php
echo dirname("c:/testweb/home.php");
echo dirname("/testweb/home.php");
?>
```

输出：

```
c:/testweb
/testweb
```

PHP disk_free_space() 函数

定义和用法

disk_free_space() 函数返回目录中的可用空间

语法

```
disk_free_space(directory)
```

参数	描述
directory	必需。规定要检查的目录。

说明

directory 参数是一个目录的字符串。该函数将根据相应的文件系统或磁盘分区返回可用的字节数。

例子

```
<?php
echo disk_free_space("C:");
?>
```

输出：

```
209693288558
```

PHP disk_total_space() 函数

定义和用法

disk_total_space() 函数返回指定目录的磁盘总大小。

语法

```
disk_total_space(directory)
```

参数	描述
directory	必需。规定要检查的目录。

说明

directory 参数是一个目录的字符串。该函数将根据相应的文件系统或磁盘分区返回所有的字节数。

提示和注释

提示：本函数返回的是该目录所在的磁盘分区的总大小，因此在给出同一个磁盘分区的不同目录作为参数所得到的结果完全相同。在 Unix 和 Windows 200x/XP 中都支持将一个磁盘分区加载为一个子目录，这时正确使用本函数就很有意义。

例子

```
<?php
echo disk_total_space("C:");
?>
```

输出类似这样：

```
509693888668
```

PHP diskfreespace() 函数

定义和用法

diskfreespace() 函数返回目录中的可用空间。该函数是 disk_free_space() 函数的别名。

语法

```
diskfreespace(directory)
```

参数	描述
directory	必需。规定要检查的目录。

说明

directory 参数是一个目录的字符串。该函数将根据相应的文件系统或磁盘分区返回可用的字节数。

例子

```
<?php
echo diskfreespace("C:");
?>
```

输出：

```
209693288558
```

PHP fclose() 函数

定义和用法

fclose() 函数关闭一个打开文件。

语法

```
fclose(file)
```

参数	描述
file	必需。规定要关闭的文件。

说明

file 参数是一个文件指针。fclose() 函数关闭该指针指向的文件。

如果成功则返回 true，否则返回 false。

文件指针必须有效，并且是通过 fopen() 或 fsockopen() 成功打开的。

例子

```
<?php
$file = fopen("test.txt","r");

//执行的一些代码...

fclose($file);
?>
```

PHP feof() 函数

定义和用法

feof() 函数检测是否已到达文件末尾 (eof)。

如果文件指针到了 EOF 或者出错时则返回 TRUE，否则返回一个错误（包括 socket 超时），其它情况则返回 FALSE。

语法

```
feof(file)
```

参数	描述
file	必需。规定要检查的打开文件。

说明

file 参数是一个文件指针。这个文件指针必须有效，并且必须指向一个由 fopen() 或 fsockopen() 成功打开（但还没有被 fclose() 关闭）的文件。

提示和注释

提示：feof() 函数对遍历长度未知的数据很有用。

注意：如果服务器没有关闭由 fsockopen() 所打开的连接，feof() 会一直等待直到超时而返回 TRUE。默认的超时限制是 60 秒，可以使用 stream_set_timeout() 来改变这个值。

注意：如果传递的文件指针无效可能会陷入无限循环中，因为 EOF 不会返回 TRUE。

例子

```
<?php
$file = fopen("test.txt", "r");

//输出文本中所有的行，直到文件结束为止。
while(! feof($file))
  {
  echo fgets($file). "<br />";
  }

fclose($file);
?>
```

输出：

```
Hello, this is a test file.
There are three lines here.
This is the last line.
```

PHP fflush() 函数

定义和用法

fflush() 函数将缓冲内容输出到文件。

语法

```
fflush(file)
```

参数	描述
file	必需。规定要检查的文件流。

说明

本函数强制将所有缓冲的输出写入 *file* 文件句柄所指向的资源。如果成功则返回 true，否则返回 false。

文件指针必须有效，并且必须指向一个由 fopen() 或 fsockopen() 成功打开（但还没有被 fclose() 关闭）的文件。

例子

```
<?php
file = fopen("test.txt","r+");

// 一些代码

fflush($file);
?>
```

PHP fgetc() 函数

定义和用法

fgetc() 函数从文件指针中读取一个字符。

语法

```
fgetc(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

返回一个包含有一个字符的字符串，该字符从 *file* 指向的文件中得到。碰到 EOF 则返回 false。

文件指针必须有效，并且必须指向一个由 fopen() 或 fsockopen() 成功打开（但还没有被 fclose() 关闭）的文件。

提示和注释

注意：该函数可能返回布尔值 false，但也可能返回一个与 false 等值的非布尔值，例如 0 或者 ""。

注释：该函数可安全用于二进制对象。

例子

例子 1

```
<?php

$file = fopen("test.txt","r");
echo fgetc($file);
fclose($file);

?>
```

输出类似：

```
H
```

例子 2

```
<?php

$file = fopen("test.txt","r");

while (! feof ($file))
  {
  echo fgetc($file);
  }

fclose($file);
?>
```

输出类似：

```
Hello, this is a test file.
```

PHP fgetcsv() 函数

定义和用法

fgetcsv() 函数从文件指针中读入一行并解析 CSV 字段。

与 fgets() 类似，不同的是 fgetcsv() 解析读入的行并找出 CSV 格式的字段，然后返回一个包含这些字段的数组。

fgetcsv() 出错时返回 FALSE，包括碰到文件结束时。

注释：从 PHP 4.3.5 起，fgetcsv() 的操作是二进制安全的。

语法

```
fgetcsv(file,length,separator,enclosure)
```

参数	描述
file	必需。规定要检查的文件。
length	可选。规定行的最大长度。必须大于 CVS 文件内最长的一行。在 PHP 5 中该参数是可选的。在 PHP 5 之前是必需的。如果忽略（在 PHP 5.0.4 以后的版本中设为 0）该参数的话，那么长度就没有限制，不过可能会影响执行效率。
separator	可选。设置字段分界符（只允许一个字符），默认值为逗号。
enclosure	可选。设置字段环绕符（只允许一个字符），默认值为双引号。该参数是在 PHP 4.3.0 中添加的。

提示和注释

注释：CSV 文件中的空行将被返回为一个包含有单个 null 字段的数组，不会被当成错误。

注释：该函数对区域设置是敏感的。比如说 LANG 设为 en_US.UTF-8 的话，单字节编码的文件就会出现读取错误。

注释：如果碰到 PHP 在读取文件时不能识别 Macintosh 文件的行结束符，可以激活 auto_detect_line_endings 运行时配置选项。

例子

例子 1

```
<?php

$file = fopen("contacts.csv","r");
print_r(fgetcsv($file));
fclose($file);

?>
```

CSV 文件：

```
George, John, Thomas, USA
James, Adrew, Martin, USA
```

输出类似：

```
Array
(
[0] => George
[1] => John
[2] => Thomas
[3] => USA
)
```

例子 2

```
<?php

$file = fopen("contacts.csv","r");

while(! feof($file))
  {
  print_r(fgetcsv($file));
  }

fclose($file);

?>
```

CSV 文件：

```
George, John, Thomas, USA
James, Adrew, Martin, USA
```

输出类似：

```
Array
(
[0] => George
[1] => John
[2] => Thomas
[3] => USA

Array
(
[0] => James
[1] => Adrew
[2] => Martin
[3] => USA
)
```

PHP fgets() 函数

定义和用法

fgets() 函数从文件指针中读取一行。

语法

```
fgets(file,length)
```

参数	描述
file	必需。规定要读取的文件。
length	可选。规定要读取的字节数。默认是 1024 字节。

说明

从 *file* 指向的文件中读取一行并返回长度最多为 *length* - 1 字节的字符串。碰到换行符（包括在返回值中）、EOF 或者已经读取了 *length* - 1 字节后停止（要看先碰到那一种情况）。如果没有指定 *length*，则默认为 1K，或者说 1024 字节。

若失败，则返回 false。

提示和注释

注释：*length* 参数从 PHP 4.2.0 起成为可选项，如果忽略，则行的长度被假定为 1024 字节。从 PHP 4.3 开始，忽略掉 *length* 将继续从流中读取数据直到行结束。如果文件中的大多数行都大于 8 KB，则在脚本中指定最大行的长度在利用资源上更为有效。

注释：从 PHP 4.3 开始本函数可以安全用于二进制文件。早期的版本则不行。

注释：如果碰到 PHP 在读取文件时不能识别 Macintosh 文件的行结束符，可以激活 auto_detect_line_endings 运行时配置选项。

例子

例子 1

```
<?php

$file = fopen("test.txt","r");
echo fgets($file);
fclose($file);

?>
```

输出类似：

```
Hello, this is a test file.
```

例子 2

```
<?php

$file = fopen("test.txt","r");

while(! feof($file))
  {
  echo fgets($file). "<br />";
  }

fclose($file);
?>
```

输出类似：

```
Hello, this is a test file.
There are three lines here.
This is the last line.
```

PHP fgetss() 函数

定义和用法

fgetss() 函数从打开的文件中读取一行并过滤掉 HTML 和 PHP 标记。

与 fgets() 相同，不同的是 fgetss 尝试从读取的文本中去掉任何 HTML 和 PHP 标记。

语法

```
fgetss(_file_,_length_,_tags_)
```

参数	描述
file	必需。规定要读取的文件。
length	可选。规定要读取的字节数。默认是 1024 字节。该参数在 PHP 5 之前是必需的。
tags	可选。规定不会被删除的标签。

说明

可以用可选的第三个参数 *tags* 指定哪些标记不被去掉。

若失败，则返回 false。

例子

例子 1

```
<?php

$file = fopen("test.htm","r");
echo fgetss($file);
fclose($file);

?>
```

输出类似：

```
This is a paragraph.
```

例子 2

```
<?php

$file = fopen("test.htm","r");
echo fgetss($file,1024,"<p>,<b>");
fclose($file);

?>
```

输出类似：

```
**This is a paragraph.**
```

输出的源代码是：

```
<p><b>This is a paragraph.</b></p>
```

PHP file() 函数

定义和用法

file() 函数把整个文件读入一个数组中。

与 file_get_contents() 类似，不同的是 file() 将文件作为一个数组返回。数组中的每个单元都是文件中相应的一行，包括换行符在内。

如果失败，则返回 false。

语法

```
file(path,include_path,context)
```

参数	描述
path	必需。规定要读取的文件。
include_path	可选。如果也想在 include_path 中搜寻文件的话，可以将该参数设为 "1"。
context	可选。规定文件句柄的环境。context 是一套可以修改流的行为的选项。若使用 null，则忽略。

说明

对 *context* 的支持是 PHP 5.0.0 添加的。

返回的数组中每一行都包括了行结束符，因此如果不需要行结束符时还需要使用 rtrim() 函数。

提示和注释

注释：从 PHP 4.3.0 开始，可以用 file_get_contents() 来将文件读入到一个字符串并返回。

注释：从 PHP 4.3.0 开始，file() 可以安全用于二进制文件。

注释：如果碰到 PHP 在读取文件时不能识别 Macintosh 文件的行结束符，可以激活 auto_detect_line_endings 运行时配置选项。

例子

```
<?php
print_r(file("test.txt"));
?>
```

输出：

```
Array
(
[0] => Hello World. Testing testing!
[1] => Another day, another line.
[2] => If the array picks up this line,
[3] => then is it a pickup line?
)
```

PHP file_exists() 函数

定义和用法

file_exists() 函数检查文件或目录是否存在。

如果指定的文件或目录存在则返回 true，否则返回 false。

语法

```
file_exists(path)
```

参数	描述
path	必需。规定要检查的路径。

例子

```
<?php
echo file_exists("test.txt");
?>
```

输出：

```
1
```

PHP file_get_contents() 函数

定义和用法

file_get_contents() 函数把整个文件读入一个字符串中。

和 file() 一样，不同的是 file_get_contents() 把文件读入一个字符串。

file_get_contents() 函数是用于将文件的内容读入到一个字符串中的首选方法。如果操作系统支持，还会使用内存映射技术来增强性能。

语法

```
file_get_contents(_path_,_include_path_,_context_,_start_,_max_length_)
```

参数	描述
path	必需。规定要读取的文件。
include_path	可选。如果也想在 include_path 中搜寻文件的话，可以将该参数设为 "1"。
context	可选。规定文件句柄的环境。context 是一套可以修改流的行为的选项。若使用 null，则忽略。
start	可选。规定在文件中开始读取的位置。该参数是 PHP 5.1 新加的。
max_length	可选。规定读取的字节数。该参数是 PHP 5.1 新加的。

说明

对 *context* 参数的支持是 PHP 5.0.0 添加的。

提示和注释

注释：本函数可安全用于二进制对象。

例子

```
<?php
echo file_get_contents("test.txt");
?>
```

输出：

```
This is a test file with test text.
```

PHP file_put_contents() 函数

定义和用法

file_put_contents() 函数把一个字符串写入文件中。

与依次调用 fopen()，fwrite() 以及 fclose() 功能一样。

语法

```
file_put_contents(file,data,mode,context)
```

参数	描述
file	必需。规定要写入数据的文件。如果文件不存在，则创建一个新文件。
data	可选。规定要写入文件的数据。可以是字符串、数组或数据流。
mode	可选。规定如何打开/写入文件。可能的值：`FILE_USE_INCLUDE_PATH` `FILE_APPEND` `LOCK_EX`
context	可选。规定文件句柄的环境。context 是一套可以修改流的行为的选项。若使用 null，则忽略。

说明

参数 *data* 可以是数组（但不能是多维数组）。

自 PHP 5.1.0 起，*data* 参数也可以被指定为 stream 资源，stream 中所保存的缓存数据将被写入到指定文件中，这种用法就相似于使用 stream_copy_to_stream() 函数。

对 *context* 参数的支持是 PHP 5.0.0 添加的。

返回值

该函数将返回写入到文件内数据的字节数。

提示和注释

提示：使用 FILE_APPEND 可避免删除文件中已有的内容。

注释：本函数可安全用于二进制对象。

例子

```
<?php
echo file_put_contents("test.txt","Hello World. Testing!");
?>
```

输出：

```
26
```

PHP fileatime() 函数

定义和用法

fileatime() 函数返回指定文件的上次访问时间。

该函数返回文件上次被访问的时间。如果出错则返回 false。时间以 Unix 时间戳的方式返回。

语法

```
fileatime(filename)
```

参数	描述
filename	必需。规定要检查的文件。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

注释：文件的 atime 应该在不论何时读取了该文件中的数据块时被更改。当一个应用程序定期访问大量文件或目录时很影响性能。有些 Unix 文件系统可以在加载时关闭 atime 的更新以提高这类程序的性能。USENET 新闻组假脱机是一个常见的例子。在这种文件系统下，本函数没有用处。

例子

```
<?php
echo fileatime("test.txt");
echo "Last access: ".date("F d Y H:i:s.",fileatime("test.txt"));
?>
```

输出：

```
1140684501
Last access: February 23 2006 09:48:21.
```

PHP filectime() 函数

定义和用法

filectime() 函数返回指定文件的上次 inode 修改时间。

该函数返回文件上次 inode 被修改的时间。如果出错则返回 false。时间以 Unix 时间戳的方式返回。

语法

```
fileatime(filename)
```

参数	描述
filename	必需。规定要检查的文件。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

注意：在大多数 Unix 文件系统中，当一个文件的 inode 数据被改变时则该文件被认为是修改了。也就是说，当文件的权限，所有者，所有组或其它 inode 中的元数据被更新时。参见 filemtime()（这才是你想用于在 Web 页面中建立“最后更新时间”脚注的函数）和 fileatime()。

注释：某些 Unix 说明文本中把 ctime 说成是该文件建立的时间，这是错的。在大多数 Unix 文件系统中，没有 Unix 文件的建立时间。

例子

```
<?php
echo filectime("test.txt");
echo "Last change: ".date("F d Y H:i:s.",filectime("test.txt"));
?>
```

输出：

```
1138609592
Last change: January 30 2006 09:26:32.
```

PHP filegroup() 函数

定义和用法

filegroup() 函数返回指定文件的组 ID。

若成功，则返回指定文件所属组的 ID。若失败，则返回 false 以及一个 E_WARNING 级别的错误。

组 ID 以数字格式返回。

语法

```
filegroup(filename)
```

参数	描述
filename	必需。规定要检查的文件。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

提示：请使用 posix_getgrgid() 来将组 ID 转换为组名。

例子

```
<?php
echo filegroup("test.txt");
?>
```

PHP fileinode() 函数

定义和用法

fileinode() 函数返回文件的 inode 编号。

若成功，则返回指定文件的 inode 节点号。若失败，则返回 false。

语法

```
fileinode(filename)
```

参数	描述
filename	必需。规定要检查的文件。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
echo fileinode("test.txt");
?>
```

PHP filemtime() 函数

定义和用法

filemtime() 函数返回文件内容上次的修改时间。

若成功，则时间以 Unix 时间戳的方式返回。若失败，则返回 false。

语法

```
filemtime(filename)
```

参数	描述
filename	必需。规定要检查的文件。

说明

本函数返回文件中的数据块上次被写入的时间，也就是说，文件的内容上次被修改的时间。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
echo filemtime("test.txt");
echo "Last modified: ".date("F d Y H:i:s.",filemtime("test.txt"));
?>
```

输出：

```
1139919766
Last modified: February 14 2006 13:22:46.
```

PHP fileowner() 函数

定义和用法

fileowner() 函数返回文件的所有者。

若成功，则返回文件所有的用户 ID。若失败，则返回 false。用户 ID 以数字格式返回。

语法

```
fileowner(filename)
```

参数	描述
filename	必需。规定要检查的文件。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

提示：用户 ID 以数字格式返回，请使用 posix_getpwuid() 来把用户 ID 转换为用户名。

例子

```
<?php
echo fileowner("test.txt");
?>
```

PHP fileperms() 函数

定义和用法

fileperms() 函数返回文件或目录的权限。

若成功，则返回文件的访问权限。若失败，则返回 false。

语法

```
fileperms(filename)
```

参数	描述
filename	必需。规定要检查的文件。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

例子 1

```
<?php
echo fileperms("test.txt");
?>
```

输出：

```
33206
```

例子 2

以八进制值返回权限：

```
<?php
echo substr(sprintf("%o",fileperms("test.txt")),-4);
?>
```

输出：

```
1777
```

PHP filesize() 函数

定义和用法

filesize() 函数返回指定文件的大小。

若成功，则返回文件大小的字节数。若失败，则返回 false 并生成一条 E_WARNING 级的错误。

语法

```
filesize(filename)
```

参数	描述
filename	必需。规定要检查的文件。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
echo filesize("test.txt");
?>
```

输出：

```
20
```

PHP filetype() 函数

定义和用法

filetype() 函数返回指定文件或目录的类型。

若成功，则返回 7 种可能的值。若失败，则返回 false。

可能的值：

- fifo
- char
- dir
- block
- link
- file
- unknown

语法

```
filetype(filename)
```

参数	描述
filename	必需。规定要检查的文件。

提示和注释

提示：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

例子 1

```
<?php
echo filetype("test.txt");
?>
```

输出：

```
file
```

例子 2

```
<?php
echo filetype("images");
?>
```

输出：

```
dir
```

PHP flock() 函数

定义和用法

flock() 函数锁定或释放文件。

若成功，则返回 true。若失败，则返回 false。

语法

```
flock(file,lock,block)
```

参数	描述
file	必需。规定要锁定或释放的已打开的文件。
lock	必需。规定要使用哪种锁定类型。
block	可选。若设置为 1 或 true，则当进行锁定时阻挡其他进程。

说明

flock() 操作的 *file* 必须是一个已经打开的文件指针。

lock 参数可以是以下值之一：

- 要取得共享锁定（读取的程序），将 *lock* 设为 LOCK_SH（PHP 4.0.1 以前的版本设置为 1）。
- 要取得独占锁定（写入的程序），将 *lock* 设为 LOCK_EX（PHP 4.0.1 以前的版本中设置为 2）。
- 要释放锁定（无论共享或独占），将 *lock* 设为 LOCK_UN（PHP 4.0.1 以前的版本中设置为 3）。
- 如果不希望 flock() 在锁定时堵塞，则给 *lock* 加上 LOCK_NB（PHP 4.0.1 以前的版本中设置为 4）。

提示和注释

提示：可以通过 fclose() 来释放锁定操作，代码执行完毕时也会自动调用。

注释：由于 flock() 需要一个文件指针， 因此可能不得不用一个特殊的锁定文件来保护打算通过写模式打开的文件的访问（在 fopen() 函数中加入 "w" 或 "w+"）。

例子

```
<?php

$file = fopen("test.txt","w+");

// 排它性的锁定
if (flock($file,LOCK_EX))
  {
  fwrite($file,"Write something");
  // release lock
  flock($file,LOCK_UN);
  }
else
  {
  echo "Error locking file!";
  }

fclose($file);
?>
```

PHP fnmatch() 函数

定义和用法

fnmatch() 函数根据指定的模式来匹配文件名或字符串。

语法

```
fnmatch(pattern,string,flags)
```

参数	描述
pattern	必需。规定要检索的模式。
string	必需。规定要检查的字符串或文件。
flags	可选。

说明

此函数对于文件名尤其有用，但也可以用于普通的字符串。普通用户可能习惯于 shell 模式或者至少其中最简单的形式 '?' 和 '*' 通配符，因此使用 fnmatch() 来代替 ereg() 或者 preg_match() 来进行前端搜索表达式输入对于非程序员用户更加方便。

提示和注释

重要事项：目前该函数无法在 Windows 或其它非 POSIX 兼容的系统上使用。

例子

根据 shell 通配符来检查颜色名：

```
<?php
$txt = "My car is darkgrey..."
if (fnmatch("*gr[ae]y",$txt))
  {
  echo "some form of gray ...";
  }
?>
```

PHP fopen() 函数

定义和用法

fopen() 函数打开文件或者 URL。

如果打开失败，本函数返回 FALSE。

语法

```
fopen(filename,mode,include_path,context)
```

参数	描述
filename	必需。规定要打开的文件或 URL。
mode	必需。规定要求到该文件/流的访问类型。可能的值见下表。
include_path	可选。如果也需要在 include_path 中检索文件的话，可以将该参数设为 1 或 TRUE。
context	可选。规定文件句柄的环境。Context 是可以修改流的行为的一套选项。

mode 参数的可能的值

mode	说明
"r"	只读方式打开，将文件指针指向文件头。
"r+"	读写方式打开，将文件指针指向文件头。
"w"	写入方式打开，将文件指针指向文件头并将文件大小截为零。如果文件不存在则尝试创建之。
"w+"	读写方式打开，将文件指针指向文件头并将文件大小截为零。如果文件不存在则尝试创建之。
"a"	写入方式打开，将文件指针指向文件末尾。如果文件不存在则尝试创建之。
"a+"	读写方式打开，将文件指针指向文件末尾。如果文件不存在则尝试创建之。
"x"	创建并以写入方式打开，将文件指针指向文件头。如果文件已存在，则 fopen() 调用失败并返回 FALSE，并生成一条 E_WARNING 级别的错误信息。如果文件不存在则尝试创建之。这和给底层的 open(2) 系统调用指定 O_EXCL|O_CREAT 标记是等价的。此选项被 PHP 4.3.2 以及以后的版本所支持，仅能用于本地文件。
"x+"	创建并以读写方式打开，将文件指针指向文件头。如果文件已存在，则 fopen() 调用失败并返回 FALSE，并生成一条 E_WARNING 级别的错误信息。如果文件不存在则尝试创建之。这和给底层的 open(2) 系统调用指定 O_EXCL|O_CREAT 标记是等价的。此选项被 PHP 4.3.2 以及以后的版本所支持，仅能用于本地文件。

说明

fopen() 将 *filename* 指定的名字资源绑定到一个流上。如果 *filename* 是 "*scheme://...*" 的格式，则被当成一个 URL，PHP 将搜索协议处理器（也被称为封装协议）来处理此模式。如果该协议尚未注册封装协议，PHP 将发出一条消息来帮助检查脚本中潜在的问题并将 *filename* 当成一个普通的文件名继续执行下去。

如果 PHP 认为 *filename* 指定的是一个本地文件，将尝试在该文件上打开一个流。该文件必须是 PHP 可以访问的，因此需要确认文件访问权限允许该访问。如果激活了安全模式或者 open_basedir 则会应用进一步的限制。

如果 PHP 认为 *filename* 指定的是一个已注册的协议，而该协议被注册为一个网络 URL，PHP 将检查并确认 allow_url_fopen 已被激活。如果关闭了，PHP 将发出一个警告，而 fopen 的调用则失败。

对 *context* 的支持是 PHP 5.0.0 添加的。

提示和注释

注释：不同的操作系统家族具有不同的行结束习惯。当写入一个文本文件并想插入一个新行时，需要使用符合操作系统的行结束符号。基于 Unix 的系统使用 \n 作为行结束字符，基于 Windows 的系统使用 \r\n 作为行结束字符，基于 Macintosh 的系统使用 \r 作为行结束字符。如果写入文件时使用了错误的行结束符号，则其它应用程序打开这些文件时可能会表现得很怪异。

Windows 下提供了一个文本转换标记（"t"）可以透明地将 \n 转换为 \r\n。与此对应还可以使用 "b" 来强制使用二进制模式，这样就不会转换数据。要使用这些标记，要么用 "b" 或者用 "t" 作为 mode 参数的最后一个字符。

默认的转换模式依赖于 SAPI 和所使用的 PHP 版本，因此为了便于移植鼓励总是指定恰当的标记。如果是操作纯文本文件并在脚本中使用了 \n 作为行结束符，但还要期望这些文件可以被其它应用程序例如 Notepad 读取，则在 mode 中使用 "t"。在所有其它情况下使用 "b"。

在操作二进制文件时如果没有指定 "b" 标记，可能会碰到一些奇怪的问题，包括坏掉的图片文件以及关于 \r\n 字符的奇怪问题。

注释：为移植性考虑，强烈建议在用 fopen() 打开文件时总是使用 "b" 标记。

注释：再一次，为移植性考虑，强烈建议你重写那些依赖于 "t" 模式的代码使其使用正确的行结束符并改成 "b" 模式。

例子

```
<?php
$file = fopen("test.txt","r");
$file = fopen("/home/test/test.txt","r");
$file = fopen("/home/test/test.gif","wb");
$file = fopen("http://www.example.com/","r");
$file = fopen("ftp://user:password@example.com/test.txt","w");
?>
```

PHP fpassthru() 函数

定义和用法

fpassthru() 函数输出文件指针处的所有剩余数据。

该函数将给定的文件指针从当前的位置读取到 EOF，并把结果写到输出缓冲区。

语法

```
fpassthru(file)
```

参数	描述
file	必需。规定要读取的打开文件或资源。

说明

如果发生错误， fpassthru() 返回 false。否则 fpassthru() 返回从 *file* 读取并传递到输出的字符数目。

文件指针必须有效，并且必须指向一个由 fopen() 或 fsockopen() 成功打开（但还没有被 fclose() 关闭）的文件。

提示和注释

提示：如果已经向文件写入数据，就必须调用 rewind() 来将文件指针指向文件头。

提示：如果既不修改文件也不在特定位置检索，只想将文件的内容下载到输出缓冲区，应该使用 readfile()，这样可以省去 fopen() 调用。

注释：当在 Windows 系统中将 fpassthru() 用于二进制文件时，要确保在用 fopen() 打开文件时在 mode 中附加了 b 来将文件以二进制方式打开。鼓励在处理二进制文件时使用 b 标志，即使系统并不需要，这样可以使脚本的移植性更好。

例子

例子 1

```
<?php
$file = fopen("test.txt","r");

// 读取第一行
fgets($file);

// 把文件的其余部分发送到输出缓存
echo fpassthru($file);

fclose($file);
?>
```

输出：

```
There are three lines in this file.
This is the last line.59
```

注：59 指示被传递的字符数。

例子 2

转储 www 服务器的 index 页：

```
<?php
$file = fopen("http://www.example.com","r");
fpassthru($file);
?>
```

PHP fputcsv() 函数

定义和用法

fputcsv() 函数将行格式化为 CSV 并写入一个打开的文件。

该函数返回写入字符串的长度。若出错，则返回 false。。

语法

```
fputcsv(file,fields,seperator,enclosure)
```

参数	描述
file	必需。规定要写入的打开文件。
fields	必需。规定要从中获得数据的数组。
seperator	可选。规定字段分隔符的字符。默认是逗号 (,)。
enclosure	可选。规定字段环绕符的字符。默认是双引号 "。

说明

fputcsv() 将一行（用 *fields* 数组传递）格式化为 CSV 格式并写入由 *file* 指定的文件。

提示和注释

提示：参见 fgetcsv() 函数。

例子

```
<?php
$list = array
(
"George,John,Thomas,USA",
"James,Adrew,Martin,USA",
);

$file = fopen("contacts.csv","w");

foreach ($list as $line)
  {
  fputcsv($file,split(',',$line));
  }

fclose($file);
?>
```

以上代码执行后，CSV 文件会类似这样：

```
George,John,Thomas,USA
James,Adrew,Martin,USA
```

PHP fputs() 函数

定义和用法

fputs() 函数写入文件（可安全用于二进制文件）。

fputs() 函数是 fwrite() 函数的别名。

语法

```
fputs(file,string,length)
```

参数	描述
file	必需。规定要写入的打开文件。
string	必需。规定要写入文件的字符串。
length	可选。规定要写入的最大字节数。

说明

fwrite() 把 *string* 的内容写入文件指针 *file* 处。 如果指定了 *length*，当写入了 *length* 个字节或者写完了 *string* 以后，写入就会停止，视乎先碰到哪种情况。

fwrite() 返回写入的字符数，出现错误时则返回 false。

例子

```
<?php
$file = fopen("test.txt","w");
echo fputs($file,"Hello World. Testing!");
fclose($file);
?>
```

输出：

```
21
```

PHP fread() 函数

定义和用法

fread() 函数读取文件（可安全用于二进制文件）。

语法

```
fread(file,length)
```

参数	描述
file	必需。规定要读取打开文件。
length	必需。规定要读取的最大字节数。

说明

fread() 从文件指针 *file* 读取最多 *length* 个字节。该函数在读取完最多 *length* 个字节数，或到达 EOF 的时候，或（对于网络流）当一个包可用时，或（在打开用户空间流之后）已读取了 8192 个字节时就会停止读取文件，视乎先碰到哪种情况。

返回所读取的字符串，如果出错返回 false。

提示和注释

提示：如果只是想将一个文件的内容读入到一个字符串中，请使用 file_get_contents()，它的性能比 fread() 好得多。

例子

例子 1

从文件中读取 10 个字节：

```
<?php
$file = fopen("test.txt","r");
fread($file,"10");
fclose($file);
?>
```

例子 2

读取整个文件：

```
<?php
$file = fopen("test.txt","r");
fread($file,filesize("test.txt"));
fclose($file);
?>
```

PHP fscanf() 函数

定义和用法

fscanf() 函数根据指定的格式对来自打开的文件的输入进行解析。

语法

```
fscanf(file,format,mixed)
```

参数	描述
file	必需。规定要检查的文件。
format	必需。规定格式。
mixed	可选。

说明

fscanf() 函数与 sscanf() 相似，但是它从与 *file* 关联的文件中接受输入并根据指定的 *format* 来解释输入。如果只给此函数传递了两个参数，解析后的值会被作为数组返回。否则，如果提供了可选参数，此函数将返回被赋值的数目。可选参数必须用引用传递。

提示和注释

注释：格式字符串中的任何空白会与输入流中的任何空白匹配。这意味着甚至格式字符串中的制表符 \t 也会与输入流中的一个空格字符匹配。

注释：在 PHP 4.3.0 之前，从文件中读入的最大字符数是 512（或者第一个 \n，看先碰到哪种情况）。从 PHP 4.3.0 起可以读取任意长的行。

PHP fseek() 函数

定义和用法

fseek() 函数在打开的文件中定位。

该函数把文件指针从当前位置向前或向后移动到新的位置，新位置从文件头开始以字节数度量。

成功则返回 0；否则返回 -1。注意，移动到 EOF 之后的位置不会产生错误。

语法

```
fseek(file,offset,whence)
```

参数	描述
file	必需。规定要在其中定位的文件。
offset	必需。规定新的位置（从文件头开始以字节数度量）。
whence	可选。可能的值： `SEEK_SET` - 设定位置等于 *offset* 字节。默认。 `SEEK_CUR` - 设定位置为当前位置加上 *offset*。 `SEEK_END` - 设定位置为文件末尾加上 *offset* （要移动到文件尾之前的位置，*offset* 必须是一个负值）。

说明

whence 参数是 PHP 4.0.0 之后增加的。

提示和注释

提示：通过使用 ftell() 来找到当前位置。

例子

```
<?php
$file = fopen("test.txt","r");

// 读取第一行
fgets($file);

// 倒回文件的开头
fseek($file,0);
?>
```

PHP fstat() 函数

定义和用法

fstat() 函数返回关于打开文件的信息。

语法

```
fstat(file)
```

参数	描述
pipe	必需。规定要检查的打开文件。

说明

获取由文件指针 handle 所打开文件的统计信息。

该函数返回的数组具有该文件的统计信息，该数组包含以下元素：

数字下标	关联键名（自 **PHP 4.0.6**）	说明
0	dev	设备名
1	ino	号码
2	mode	inode 保护模式
3	nlink	被连接数目
4	uid	所有者的用户 id
5	gid	所有者的组 id
6	rdev	设备类型，如果是 inode 设备的话
7	size	文件大小的字节数
8	atime	上次访问时间（Unix 时间戳）
9	mtime	上次修改时间（Unix 时间戳）
10	ctime	上次改变时间（Unix 时间戳）
11	blksize	文件系统 IO 的块大小
12	blocks	所占据块的数目

提示和注释

提示：本函数与 stat() 函数相似，不同的是，它是作用于已打开的文件指针而不是文件名。

例子

```
<?php
$file = fopen("test.txt","r");
print_r(fstat($file));
fclose($file);
?>
```

输出类似：

```
Array
(
[0] => 0
[1] => 0
[2] => 33206
[3] => 1
[4] => 0
[5] => 0
[6] => 0
[7] => 92
[8] => 1141633430
[9] => 1141298003
[10] => 1138609592
[11] => -1
[12] => -1
[dev] => 0
[ino] => 0
[mode] => 33206
[nlink] => 1
[uid] => 0
[gid] => 0
[rdev] => 0
[size] => 92
[atime] => 1141633430
[mtime] => 1141298003
[ctime] => 1138609592
[blksize] => -1
[blocks] => -1
)
```

PHP ftell() 函数

定义和用法

ftell() 函数在打开文件中的当前位置。

该函数返回文件指针的当前位置。若失败，则返回 false。

语法

```
ftell(file)
```

参数	描述
file	必需。规定要检查的已打开文件。

说明

文件指针 *file* 必须是有效的，且必须指向一个通过 fopen() 或 popen() 成功打开的文件。

在附加模式（加参数 "a" 打开文件）中 ftell() 会返回未定义错误。

例子

```
<?php
$file = fopen("test.txt","r");

// 输出当前位置
echo ftell($file);

// 改变当前位置
fseek($file,"15");

// 再次输出当前位置
echo ftell($file);

fclose($file);
?>
```

输出：

```
0
15
```

PHP ftruncate() 函数

定义和用法

ftruncate() 函数把文件截断到指定的长度。

语法

```
ftruncate(file,size)
```

参数	描述
file	必需。规定要截断的打开文件。
size	必需。规定新的文件大小。

说明

接受文件指针 *file* 作为参数，并将文件大小截取为 *size*。如果成功则返回 TRUE，否则返回 FALSE。

提示和注释

注释：文件只会在 append 模式下改变。在 write 模式下，必须加上 fseek() 操作。

注释：在 PHP 4.3.3 之前，ftruncate() 在成功时返回一个整数值 1，而不是布尔值的 TRUE。

例子

```
<?php
//检查文件大小
echo filesize("test.txt");
echo "<br />";

$file = fopen("test.txt", "a+");
ftruncate($file,100);
fclose($file);

//清空缓存，再次检查文件大小
clearstatcache();
echo filesize("test.txt");
?>
```

输出类似：

```
792
100
```

PHP fwrite() 函数

定义和用法

fwrite() 函数写入文件（可安全用于二进制文件）。

语法

```
fwrite(file,string,length)
```

参数	描述
file	必需。规定要写入的打开文件。
string	必需。规定要写入文件的字符串。
length	可选。规定要写入的最大字节数。

说明

fwrite() 把 *string* 的内容写入文件指针 *file* 处。 如果指定了 *length*，当写入了 *length* 个字节或者写完了 *string* 以后，写入就会停止，视乎先碰到哪种情况。

fwrite() 返回写入的字符数，出现错误时则返回 false。

例子

```
<?php
$file = fopen("test.txt","w");
echo fwrite($file,"Hello World. Testing!");
fclose($file);
?>
```

输出：

```
21
```

PHP glob() 函数

定义和用法

glob() 函数返回匹配指定模式的文件名或目录。

该函数返回一个包含有匹配文件 / 目录的数组。如果出错返回 false。

语法

```
glob(pattern,flags)
```

参数	描述
file	必需。规定检索模式。
size	可选。规定特殊的设定。 `GLOB_MARK` - 在每个返回的项目中加一个斜线 `GLOB_NOSORT` - 按照文件在目录中出现的原始顺序返回（不排序） `GLOB_NOCHECK` - 如果没有文件匹配则返回用于搜索的模式 `GLOB_NOESCAPE` - 反斜线不转义元字符 `GLOB_BRACE` - 扩充 {a,b,c} 来匹配 'a'，'b' 或 'c' `GLOB_ONLYDIR` - 仅返回与模式匹配的目录项 `GLOB_ERR` - 停止并读取错误信息（比如说不可读的目录），默认的情况下忽略所有错误 注释：GLOB_ERR 是 PHP 5.1 添加的。

例子

例子 1

```
<?php
print_r(glob("*.txt"));
?>
```

输出类似：

```
Array
(
[0] => target.txt
[1] => source.txt
[2] => test.txt
[3] => test2.txt
)
```

例子 2

```
<?php
print_r(glob("*.*"));
?>
```

输出类似：

```
Array
(
[0] => contacts.csv
[1] => default.php
[2] => target.txt
[3] => source.txt
[4] => tem1.tmp
[5] => test.htm
[6] => test.ini
[7] => test.php
[8] => test.txt
[9] => test2.txt
)
```

PHP is_dir() 函数

定义和用法

is_dir() 函数检查指定的文件是否是目录。

语法

```
is_dir(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

如果文件名存在并且为目录，则返回 true。如果 *file* 是一个相对路径，则按照当前工作目录检查其相对路径。

提示和注释

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
$file = "images";
if(is_dir($file))
  {
  echo ("$file is a directory");
  }
else
  {
  echo ("$file is not a directory");
  }
?>
```

输出：

```
images is a directory
```

PHP is_executable() 函数

定义和用法

is_executable() 函数检查指定的文件是否可执行。

语法

```
is_executable(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

如果文件存在且可执行，则返回 true。

提示和注释

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

注释：is_executable() 自 PHP 5.0.0 版起可用于 Windows。

例子

```
<?php
$file = "setup.exe";
if(is_executable($file))
  {
  echo ("$file is executable");
  }
else
  {
  echo ("$file is not executable");
  }
?>
```

输出：

```
setup.exe is executable
```

PHP is_file() 函数

定义和用法

is_file() 函数检查指定的文件名是否是正常的文件。

语法

```
is_file(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

如果文件存在且为正常的文件，则返回 true。

提示和注释

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

例子 1

```
<?php
$file = "test.txt";
if(is_file($file))
  {
  echo ("$file is a regular file");
  }
else
  {
  echo ("$file is not a regular file");
  }
?>
```

输出：

```
test.txt is a regular file
```

例子 2

```
<?php
var_dump(is_file('a_file.txt')) . "\n";
var_dump(is_file('/usr/bin/')) . "\n";
?>
```

输出：

```
bool(true)
bool(false)
```

PHP is_link() 函数

定义和用法

is_link() 函数判断指定文件名是否为一个符号连接。

语法

```
is_link(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

如果文件存在并且是一个符号连接，则返回 true。

提示和注释

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
$link = "images";
if(is_link($link))
  {
  echo ("$link is a link");
  }
else
  {
  echo ("$link is not a link");
  }
?>
```

输出：

```
images is not a link
```

PHP is_readable() 函数

定义和用法

is_readable() 函数判断指定文件名是否可读。

语法

```
is_readable(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

如果由 *file* 指定的文件或目录存在并且可读，则返回 TRUE。

提示和注释

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
$file = "test.txt";
if(is_readable($file))
  {
  echo ("$file is readable");
  }
else
  {
  echo ("$file is not readable");
  }
?>
```

输出：

```
test.txt is readable
```

PHP is_uploaded_file() 函数

定义和用法

is_uploaded_file() 函数判断指定的文件是否是通过 HTTP POST 上传的。

语法

```
is_uploaded_file(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

如果 *file* 所给出的文件是通过 HTTP POST 上传的则返回 TRUE。

该函数可以用于确保恶意的用户无法欺骗脚本去访问本不能访问的文件，例如 /etc/passwd。

这种检查显得格外重要，如果上传的文件有可能会造成对用户或本系统的其他用户显示其内容的话。

提示和注释

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
$file = "test.txt";
if(is_uploaded_file($file))
  {
  echo ("$file is uploaded via HTTP POST");
  }
else
  {
  echo ("$file is not uploaded via HTTP POST");
  }
?>
```

输出：

```
test.txt is not uploaded via HTTP POST
```

PHP is_writable() 函数

定义和用法

is_writable() 函数判断指定的文件是否可写。

语法

```
is_writable(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

如果文件存在并且可写则返回 true。*file* 参数可以是一个允许进行是否可写检查的目录名。

提示和注释

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
$file = "test.txt";
if(is_writable($file))
  {
  echo ("$file is writeable");
  }
else
  {
  echo ("$file is not writeable");
  }
?>
```

输出：

```
test.txt is writeable
```

PHP is_writeable() 函数

定义和用法

is_writeable() 函数判断指定的文件是否可写。

该函数是 is_writable() 函数的别名。

语法

```
is_writeable(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

如果文件存在并且可写则返回 true。*file* 参数可以是一个允许进行是否可写检查的目录名。

提示和注释

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
$file = "test.txt";
if(is_writeable($file))
  {
  echo ("$file is writeable");
  }
else
  {
  echo ("$file is not writeable");
  }
?>
```

输出：

```
test.txt is writeable
```

PHP link() 函数

定义和用法

link() 函数建立一个硬连接。

如果成功，则返回 true，失败则返回 false。

提示：创建的连接不是 HTML 链接，而是文件系统中的连接。

语法

```
link(target,link)
```

参数	描述
target	必需。
link	必需。

提示和注释

注释：本函数不能作用于远程文件，被检查的文件必须通过服务器的文件系统访问。

注释：本函数不能工作在 windows 平台上。

PHP linkinfo() 函数

定义和用法

linkinfo() 函数返回连接的信息。

本函数返回设备 ID。若出错，则返回 0 或 FALSE。

语法

```
linkinfo(path)
```

参数	描述
path	必需。规定要检查的路径

提示和注释

注释：本函数不能工作在 windows 平台上。

PHP lstat() 函数

定义和用法

lstat() 函数返回关于文件或符号连接的信息。

语法

```
lstat(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

获取由 *file* 参数指定的文件或符号连接的统计信息。

lstat() 的返回格式

数字下标	关联键名（自 **PHP 4.0.6**）	说明
0	dev	设备名
1	ino	号码
2	mode	inode 保护模式
3	nlink	被连接数目
4	uid	所有者的用户 id
5	gid	所有者的组 id
6	rdev	设备类型，如果是 inode 设备的话
7	size	文件大小的字节数
8	atime	上次访问时间（Unix 时间戳）
9	mtime	上次修改时间（Unix 时间戳）
10	ctime	上次改变时间（Unix 时间戳）
11	blksize	文件系统 IO 的块大小
12	blocks	所占据块的数目

提示和注释

提示：本函数与 stat() 函数相同，不同之处只有一点：如果 *file* 参数是符号连接的话，则该符号连接的状态被返回，而不是该符号连接所指向的文件的状态。

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
print_r(lstat("test.txt"));
?>
```

输出类似：

```
Array
(
[0] => 0
[1] => 0
[2] => 33206
[3] => 1
[4] => 0
[5] => 0
[6] => 0
[7] => 92
[8] => 1141633430
[9] => 1141298003
[10] => 1138609592
[11] => -1
[12] => -1
[dev] => 0
[ino] => 0
[mode] => 33206
[nlink] => 1
[uid] => 0
[gid] => 0
[rdev] => 0
[size] => 92
[atime] => 1141633430
[mtime] => 1141298003
[ctime] => 1138609592
[blksize] => -1
[blocks] => -1
)
```

PHP mkdir() 函数

定义和用法

mkdir() 函数创建目录。

若成功，则返回 true，否则返回 false。

语法

```
mkdir(path,mode,recursive,context)
```

参数	描述
path	必需。规定要创建的目录的名称。
mode	必需。规定权限。默认是 0777。
recursive	必需。规定是否设置递归模式。
context	必需。规定文件句柄的环境。Context 是可修改流的行为的一套选项。

说明

mkdir() 尝试新建一个由 *path* 指定的目录。

默认的 *mode* 是 0777，意味着最大可能的访问权。

提示和注释

注释：*mode* 在 Windows 下被忽略。自 PHP 4.2.0 起成为可选项。

注释：对 *context* 的支持是 PHP 5.0.0 添加的。

注释：*recursive* 参数是 PHP 5.0.0 添加的。

例子

```
<?php
mkdir("testing");
?>
```

PHP move_uploaded_file() 函数

定义和用法

move_uploaded_file() 函数将上传的文件移动到新位置。

若成功，则返回 true，否则返回 false。

语法

```
move_uploaded_file(file,newloc)
```

参数	描述
file	必需。规定要移动的文件。
newloc	必需。规定文件的新位置。

说明

本函数检查并确保由 *file* 指定的文件是合法的上传文件（即通过 PHP 的 HTTP POST 上传机制所上传的）。如果文件合法，则将其移动为由 *newloc* 指定的文件。

如果 *file* 不是合法的上传文件，不会出现任何操作，move_uploaded_file() 将返回 false。

如果 *file* 是合法的上传文件，但出于某些原因无法移动，不会出现任何操作，move_uploaded_file() 将返回 false，此外还会发出一条警告。

这种检查显得格外重要，如果上传的文件有可能会造成对用户或本系统的其他用户显示其内容的话。

提示和注释

注释：本函数仅用于通过 HTTP POST 上传的文件。

注意：如果目标文件已经存在，将会被覆盖。

PHP parse_ini_file() 函数

定义和用法

parse_ini_file() 函数解析一个配置文件，并以数组的形式返回其中的设置。

语法

```
parse_ini_file(file,process_sections)
```

参数	描述
file	必需。规定要检查的 ini 文件。
process_sections	可选。如果设置为 true，则返回一个多维数组，包括了配置文件中每一节的名称和设置。默认是 false。

说明

ini 文件的结构和 php.ini 的相似。

常量也可以在 ini 文件中被解析，因此如果在运行 parse_ini_file() 之前定义了常量作为 ini 的值，将会被集成到结果中去。只有 ini 的值会被求值。

由数字组成的键名和小节名会被 PHP 当作整数来处理，因此以 0 开头的数字会被当作八进制而以 0x 开头的会被当作十六进制。

提示和注释

注释：本函数可以用来读取你自己的应用程序的配置文件。本函数与 php.ini 文件没有关系，该文件在运行脚本时就已经处理过了。

注释：如果 ini 文件中的值包含任何非字母数字的字符，需要将其括在双引号中（"）。

注释：有些保留字不能作为 ini 文件中的键名，包括：null，yes，no，true 和 false。值为 null，no 和 false 等效于 ""，值为 yes 和 true 等效于 "1"。字符 {}|"~![()" 也不能用在键名的任何地方，而且这些字符在选项值中有着特殊的意义。

注释：自 PHP 5.0 版本开始，该函数也处理选项值内的新行。

例子

例子 1

"test.ini" 的内容：

```
[names]
me = Robert
you = Peter

[urls]
first = "http://www.example.com"
second = "http://www.w3school.com.cn"
```

PHP 代码：

```
<?php
print_r(parse_ini_file("test.ini"));
?>
```

输出：

```
Array
(
[me] => Robert
[you] => Peter
[first] => http://www.example.com
[second] => http://www.w3school.com.cn
)
```

例子 2

"test.ini" 的内容：

```
[names]
me = Robert
you = Peter

[urls]
first = "http://www.example.com"
second = "http://www.w3school.com.cn"
```

PHP 代码（*process_sections* 设置为 true）：

```
<?php
print_r(parse_ini_file("test.ini",true));
?>
```

输出：

```
Array
(
[names] => Array
  (
  [me] => Robert
  [you] => Peter
  )
[urls] => Array
  (
  [first] => http://www.example.com
  [second] => http://www.w3school.com.cn
  )
)
```

PHP pathinfo() 函数

定义和用法

pathinfo() 函数以数组的形式返回文件路径的信息。

语法

```
pathinfo(path,options)
```

参数	描述
path	必需。规定要检查的路径。
process_sections	可选。规定要返回的数组元素。默认是 all。可能的值： `PATHINFO_DIRNAME` - 只返回 dirname `PATHINFO_BASENAME` - 只返回 basename `PATHINFO_EXTENSION` - 只返回 extension

说明

pathinfo() 返回一个关联数组包含有 *path* 的信息。

包括以下的数组元素：

- [dirname]
- [basename]
- [extension]

提示和注释

注释：如果不是要求取得所有单元，则 pathinfo() 函数返回字符串。

例子

例子 1

```
<?php
print_r(pathinfo("/testweb/test.txt"));
?>
```

输出：

```
Array
(
[dirname] => /testweb
[basename] => test.txt
[extension] => txt
)
```

例子 2

```
<?php
print_r(pathinfo("/testweb/test.txt",PATHINFO_BASENAME));
?>
```

输出：

```
test.txt
```

PHP pclose() 函数

定义和用法

pclose() 函数关闭由 popen() 打开的管道。

语法

```
pclose(pipe)
```

参数	描述
pipe	必需。规定由 popen() 打开的管道。

说明

该函数返回运行的进程的终止状态。

若出错，则返回 false。

例子

```
<?php
$file = popen("/bin/ls","r");

//一些要执行的代码

pclose($file);
?>
```

PHP popen() 函数

定义和用法

popen() 函数打开进程文件指针。

语法

```
popen(command,mode)
```

参数	描述
command	必需。规定要执行的命令。
mode	必需。规定连接模式。 可能的值： r：只读。 w：只写 (打开并清空已有文件或创建一个新文件)

说明

打开一个指向进程的管道，该进程由派生指定的 *command* 命令执行而产生。

返回一个和 fopen() 所返回的相同的文件指针，只不过它是单向的（只能用于读或写）并且必须用 pclose() 来关闭。此指针可以用于 fgets()，fgetss() 和 fwrite()。

若出错，则返回 false。

例子

```
<?php
$file = popen("/bin/ls","r");

//一些要执行的代码

pclose($file);
?>
```

PHP readfile() 函数

定义和用法

readfile() 函数输出一个文件。

该函数读入一个文件并写入到输出缓冲。

若成功，则返回从文件中读入的字节数。若失败，则返回 false。您可以通过 @readfile() 形式调用该函数，来隐藏错误信息。

语法

```
readfile(_filename_,_include_path_,_context_)
```

参数	描述
filename	必需。规定要读取的文件。
include_path	可选。如果也想在 *include_path* 中搜索文件，可以使用该参数并将其设为 true。
context	可选。规定文件句柄的环境。*Context* 是可以修改流的行为的一套选项。

说明

对 *context* 参数的支持是 PHP 5.0.0 添加的。

提示和注释

提示：如果在 php.ini 文件中 "fopen wrappers" 已经被激活，则在本函数中可以把 URL 作为文件名来使用。

例子

```
<?php
echo readfile("test.txt");
?>
```

输出：

```
There are two lines in this file.
This is the last line.
57
```

PHP readlink() 函数

定义和用法

readlink() 函数返回符号连接指向的目标。

若成功，则该函数返回连接的目标。若失败，则返回 false。

语法

```
readlink(linkpath)
```

参数	描述
linkpath	必需。规定要检查的连接路径。

提示和注释

注释：本函数未在 Windows 平台下实现。。

例子

```
<?php
echo readlink("/user/testlink");
?>
```

PHP realpath() 函数

定义和用法

realpath() 函数返回绝对路径。

该函数删除所有符号连接（比如 '/./', '/../' 以及多余的 '/'），返回绝对路径名。

若失败，则返回 false。比如说文件不存在的话。

语法

```
readlink(linkpath)
```

参数	描述
linkpath	必需。规定要检查的连接路径。

说明

在 BSD 系统上，如果仅仅是 *linkpath* 不存在的话，PHP 并不会像其它系统那样返回 false。

例子

```
<?php
echo realpath("test.txt");
?>
```

输出：

```
C:\Inetpub\testweb\test.txt
```

PHP rename() 函数

定义和用法

rename() 函数重命名文件或目录。

若成功，则该函数返回 true。若失败，则返回 false。

语法

```
rename(_oldname_,_newname_,_context_)
```

参数	描述
oldname	必需。规定要重命名的文件或目录。
newname	必需。规定文件或目录的新名称。
context	可选。规定文件句柄的环境。*context* 是可修改流的行为的一套选项。

提示和注释

注释：在 PHP 4.3.3 之前，rename() 不能在基于 *nix 的系统中跨磁盘分区重命名文件。

注释：用于 *oldname* 中的封装协议必须和用于 *newname* 中的相匹配。

注释：对 *context* 的支持是 PHP 5.0.0 添加的。

例子

```
<?php
rename("images","pictures");
?>
```

PHP rewind() 函数

定义和用法

rewind() 函数将文件指针的位置倒回文件的开头。

若成功，则返回 true。若失败，则返回 false。

语法

```
rewind(file)
```

参数	描述
file	必需。规定已打开的文件。

例子

```
<?php
$file = fopen("test.txt","r");

//改变文件指针的位置
fseek($file,"15");

//把文件指针设定为 0
rewind($file);

fclose($file);
?>
```

PHP rmdir() 函数

定义和用法

rmdir() 函数删除空的目录。

若成功，则该函数返回 true。若失败，则返回 false。

语法

```
rmdir(dir,context)
```

参数	描述
dir	必需。规定要删除的目录。
context	必需。规定文件句柄的环境。Context 是可修改流的行为的一套选项。

说明

尝试删除 *dir* 所指定的目录。 该目录必须是空的，而且要有相应的权限。

提示和注释

注释：对 *context* 的支持是 PHP 5.0.0 添加的。

例子

```
<?php
$path = "images";
if(!rmdir($path))
  {
  echo ("Could not remove $path");
  }
?>
```

PHP set_file_buffer() 函数

定义和用法

set_file_buffer() 函数设置打开文件的缓冲大小。

若成功，则该函数返回 0。若失败，则返回 EOF。

语法

```
set_file_buffer(file,buffer)
```

参数	描述
file	必需。规定打开的文件。
buffer	必需。规定缓冲大小，以字节计。

提示和注释

注释：该函数是 stream_set_write_buffer() 的别名。

例子

创建无缓冲的流：

```
<?php
$file = fopen("test.txt","w");
if ($file)
  {
  set_file_buffer($file,0);
  fwrite($file,"Hello World. Testing!");
  fclose($file);
  }
?>
```

PHP stat() 函数

定义和用法

stat() 函数返回关于文件的信息。

语法

```
fstat(file)
```

参数	描述
file	必需。规定要检查的文件。

说明

获取由 *file* 指定的文件的统计信息。如果 *file* 是符号连接，则统计信息是关于被连接文件本身的，而不是符号连接。

如果出错，stat() 返回 false，并且发出一条警告。

返回的数组包含有文件的统计信息，该数组具有以下列出的单元，数组下标从零开始。除了数字索引之外，从 PHP 4.0.6 起还可以通过关联索引来访问。

stat() 的返回格式

数字下标	关联键名（自 **PHP 4.0.6**）	说明
0	dev	设备名
1	ino	号码
2	mode	inode 保护模式
3	nlink	被连接数目
4	uid	所有者的用户 id
5	gid	所有者的组 id
6	rdev	设备类型，如果是 inode 设备的话
7	size	文件大小的字节数
8	atime	上次访问时间（Unix 时间戳）
9	mtime	上次修改时间（Unix 时间戳）
10	ctime	上次改变时间（Unix 时间戳）
11	blksize	文件系统 IO 的块大小
12	blocks	所占据块的数目

提示和注释

提示：lstat() 与 stat() 类似，不同的是，它会返回符号连接的状态。

注释：本函数的结果会被缓存。请使用 clearstatcache() 来清除缓存。

例子

```
<?php
$file = fopen("test.txt","r");
print_r(stat($file));
fclose($file);
?>
```

输出类似：

```
Array
(
[0] => 0
[1] => 0
[2] => 33206
[3] => 1
[4] => 0
[5] => 0
[6] => 0
[7] => 92
[8] => 1141633430
[9] => 1141298003
[10] => 1138609592
[11] => -1
[12] => -1
[dev] => 0
[ino] => 0
[mode] => 33206
[nlink] => 1
[uid] => 0
[gid] => 0
[rdev] => 0
[size] => 92
[atime] => 1141633430
[mtime] => 1141298003
[ctime] => 1138609592
[blksize] => -1
[blocks] => -1
)
```

PHP symlink() 函数

定义和用法

symlink() 函数创建符号连接。

语法

```
link(target,link)
```

参数	描述
target	必需。
link	必需。

说明

symlink() 对于已有的 *target* 建立一个名为 *link* 的符号连接。

若成功则返回 true，失败则返回 false。

提示和注释

注释：本函数未在 Windows 平台下实现。

PHP tempnam() 函数

定义和用法

tempnam() 函数创建一个具有唯一文件名的临时文件。

若成功，则该函数返回新的临时文件名。若失败，则返回 false。

语法

```
tempnam(dir,prefix)
```

参数	描述
dir	必需。规定创建临时文件的目录。
prefix	必需。规定文件名的开头。

说明

在指定目录中建立一个具有唯一文件名的文件。如果该目录不存在，tempnam() 会在系统临时目录中生成一个文件，并返回其文件名。

在 PHP 4.0.6 之前，tempnam() 函数的行为取决于系统。在 Windows 下 TMP 环境变量会越过 dir 参数，在 Linux 下 TMPDIR 环境变量优先，而在 SVR4 下总是使用 dir 参数，如果其指向的目录存在的话。

提示和注释

注释：如果 PHP 不能在指定的 *dir* 参数中创建文件，则退回到系统默认值。

注释：本函数的行为在 4.0.3 版中改变了。也会建立一个临时文件以避免竞争情形，即有可能会在产生出作为文件名的字符串与脚本真正建立该文件之间会在文件系统中存在同名文件。注意，如果不再需要该文件则要删除此文件，不会自动删除的。

提示：参见 tmpfile()

例子

```
<?php
echo tempnam("C:\inetpub\testweb","TMP0");
?>
```

输出：

```
C:\inetpub\testweb\TMP1.tmp
```

PHP tmpfile() 函数

定义和用法

tmpfile() 函数以读写（w+）模式建立一个具有唯一文件名的临时文件。

文件会在关闭后（用 fclose()）自动被删除，或当脚本结束后。

语法

```
tmpfile()
```

提示和注释

提示：参见 tempnam()。

例子

```
<?php
$temp = tmpfile();

fwrite($temp, "Testing, testing.");

//倒回文件的开头
rewind($temp);

//从文件中读取 1k
echo fread($temp,1024);

//删除文件
fclose($temp);
?>
```

输出：

```
Testing, testing.
```

PHP touch() 函数

定义和用法

touch() 函数设置指定文件的访问和修改时间。

语法

```
touch(filename,time,atime)
```

参数	描述
filename	必需。规定要接触的文件。
time	可选。设置时间。默认是当前系统时间。
atime	可选。设置访问时间。默认是当前系统时间。

说明

尝试将由 *filename* 给出的文件的访问和修改时间设定为指定的时间。如果没有设置可选参数 *time*，则使用当前系统时间。如果给出了第三个参数 *atime*，则指定文件的访问时间会被设为 atime 。

如果成功则返回 true，失败则返回 false。

提示和注释

注释：如果文件不存在，则会被创建。

例子

```
<?php
touch("test.txt");
?>
```

PHP umask() 函数

定义和用法

umask() 函数改变当前的 umask。

umask() 将 PHP 的 umask 设定为 mask & 0777 并返回原来的 umask。当 PHP 被作为服务器模块使用时，在每个请求结束后 umask 会被恢复。

无参数调用 umask() 会返回当前的 umask。

语法

```
umask(mask)
```

参数	描述
mask	必需。规定新的权限。默认是 0777。

提示和注释

注释：在多线程的服务器上尽量避免使用这个函数。创建文件后要改变其权限最好还是使用 chmod()。使用 umask() 会导致并发程序和服务器发生不可预知的情况，因为它们使用相同的 umask。

PHP unlink() 函数

定义和用法

unlink() 函数删除文件。

若成功，则返回 true，失败则返回 false。

语法

```
unlink(filename,context)
```

参数	描述
filename	必需。规定要删除的文件。
context	可选。规定文件句柄的环境。Context 是可修改流的行为的一套选项。

提示和注释

注释：对 *context* 的支持是 PHP 5.0.0 添加的。

例子

```
<?php
$file = "test.txt";
if (!unlink($file))
  {
  echo ("Error deleting $file");
  }
else
  {
  echo ("Deleted $file");
  }
?>
```

PHP Filter 函数

PHP Filter 简介

PHP 过滤器用于对来自非安全来源的数据（比如用户输入）进行验证和过滤。

安装

filter 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP Filter 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
filter_has_var()	检查是否存在指定输入类型的变量。	5
filter_id()	返回指定过滤器的 ID 号。	5
filter_input()	从脚本外部获取输入，并进行过滤。	5
filter_input_array()	从脚本外部获取多项输入，并进行过滤。	5
filter_list()	返回包含所有得到支持的过滤器的一个数组。	5
filter_var_array()	获取多项变量，并进行过滤。	5
filter_var()	获取一个变量，并进行过滤。	5

PHP Filters

ID 名称	描述
FILTER_CALLBACK	调用用户自定义函数来过滤数据。
FILTER_SANITIZE_STRING	去除标签，去除或编码特殊字符。
FILTER_SANITIZE_STRIPPED	"string" 过滤器的别名。
FILTER_SANITIZE_ENCODED	URL-encode 字符串，去除或编码特殊字符。
	HTML 转义字符 '"<>& 以

FILTER_SANITIZE_SPECIAL_CHARS	及 ASCII 值小于 32 的字符。	
FILTER_SANITIZE_EMAIL	删除所有字符，除了字母、数字以及 !#$%&'*+-/=?^_`{	}~@.[]
FILTER_SANITIZE_URL	删除所有字符，除了字母、数字以及 $-_.+!*'(),{}	\^~[]`<>#%";/?:@&=
FILTER_SANITIZE_NUMBER_INT	删除所有字符，除了数字和 +-	
FILTER_SANITIZE_NUMBER_FLOAT	删除所有字符，除了数字、+- 以及 .,eE。	
FILTER_SANITIZE_MAGIC_QUOTES	应用 addslashes()。	
FILTER_UNSAFE_RAW	不进行任何过滤，去除或编码特殊字符。	
FILTER_VALIDATE_INT	在指定的范围以整数验证值。	
FILTER_VALIDATE_BOOLEAN	如果是 "1", "true", "on" 以及 "yes"，则返回 true，如果是 "0", "false", "off", "no" 以及 ""，则返回 false。否则返回 NULL。	
FILTER_VALIDATE_FLOAT	以浮点数验证值。	
FILTER_VALIDATE_REGEXP	根据 regexp，兼容 Perl 的正则表达式来验证值。	
FILTER_VALIDATE_URL	把值作为 URL 来验证。	
FILTER_VALIDATE_EMAIL	把值作为 e-mail 来验证。	
FILTER_VALIDATE_IP	把值作为 IP 地址来验证。	

PHP filter_has_var() 函数

定义和用法

filter_has_var() 函数检查是否存在指定输入类型的变量。

若成功，则返回 true，否则返回 false。

语法

```
filter_has_var(type, variable)
```

参数	描述
type	必需。规定要检查的类型。可能的值： `INPUT_GET` `INPUT_POST` `INPUT_COOKIE` `INPUT_SERVER` `INPUT_ENV`
variable	必需。规定要检查的变量。

例子

在本例中，输入变量 "name" 被发送到 PHP 页面：

```
<?php
if(!filter_has_var(INPUT_GET, "name"))
 {
 echo("Input type does not exist");
 }
else
 {
 echo("Input type exists");
 }
?>
```

输出类似：

```
Input type exists
```

PHP filter_id() 函数

定义和用法

filter_id() 函数返回指定过滤器的 ID 号。

若成功，则返回过滤器的 ID 号。如果该过滤器不存在，则返回 NULL。

语法

```
filter_id(filter_name)
```

参数	描述
type	必需。规定被获取 ID 号的过滤器。必须是过滤器名称（不是过滤器 ID 名）。请使用 filter_list() 函数来获取所有被支持的过滤器的名称。

例子

```
<?php
echo(filter_id("validate_email"));
?>
```

输出类似：

```
274
```

PHP filter_input() 函数

定义和用法

filter_input() 函数从脚本外部获取输入，并进行过滤。

本函数用于对来自非安全来源的变量进行验证，比如用户的输入。

本函数可从各种来源获取输入：

- INPUT_GET
- INPUT_POST
- INPUT_COOKIE
- INPUT_ENV
- INPUT_SERVER
- INPUT_SESSION (Not yet implemented)
- INPUT_REQUEST (Not yet implemented)

如果成功，则返回被过滤的数据，如果失败，则返回 false，如果 *variable* 参数未设置，则返回 NULL。

语法

```
filter_input(input_type, variable, filter, options)
```

参数	描述
input_type	必需。规定输入类型。参见上面的列表中可能的类型。
variable	规定要过滤的变量。
filter	可选。规定要使用的过滤器的 ID。默认是 FILTER_SANITIZE_STRING。请参见完整的 PHP Filter 函数参考手册，获得可能的过滤器。过滤器 ID 可以是 ID 名称 （比如 FILTER_VALIDATE_EMAIL），或 ID 号（比如 274）。
options	规定包含标志/选项的数组。检查每个过滤器可能的标志和选项。

例子

在本例中，我们使用 filter_input() 函数来过滤一个 POST 变量。所接受的 POST 变量是合法的 e-mail 地址。

```
<?php
if (!filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL))
 {
 echo "E-Mail is not valid";
 }
else
 {
 echo "E-Mail is valid";
 }
?>
```

输出类似：

```
E-Mail is valid
```

PHP filter_input_array() 函数

定义和用法

filter_input_array() 函数从脚本外部获取多项输入，并进行过滤。

本函数无需重复调用 filter_input()，对过滤多个输入变量很有用。

本函数可从各种来源获取输入：

- INPUT_GET
- INPUT_POST
- INPUT_COOKIE
- INPUT_ENV
- INPUT_SERVER
- INPUT_SESSION (Not yet implemented)
- INPUT_REQUEST (Not yet implemented)

如果成功，则返回被过滤的数据，如果失败，则返回 false。

语法

```
filter_input(input_type, args)
```

参数	描述
input_type	必需。规定输入类型。参见上面的列表中可能的类型。
args	可选。规定过滤器参数数组。合法的数组键是变量名。合法的值是过滤器 ID，或者规定过滤器、标志以及选项的数组。该参数也可以是一个单独的过滤器 ID，如果是这样，输入数组中的所有值由指定过滤器进行过滤。

提示和注释

提示：参见完整的 PHP Filter 参考手册，查看可与该函数一同使用的过滤器。

例子

在本例中，我们使用 filter_input_array() 函数来过滤三个 POST 变量。所接受的 POST 变量是姓名、年龄以及电子邮件地址：

```
<?php
$filters = array
 (
 "name" => array
  (
  "filter"=>FILTER_CALLBACK,
  "flags"=>FILTER_FORCE_ARRAY,
  "options"=>"ucwords"
  ),
 "age" => array
  (
  "filter"=>FILTER_VALIDATE_INT,
  "options"=>array
   (
   "min_range"=>1,
   "max_range"=>120
   )
  ),
 "email"=> FILTER_VALIDATE_EMAIL,
 );
print_r(filter_input_array(INPUT_POST, $filters));
?>
```

输出类似：

```
Array
 (
 [name] => Peter
 [age] => 41
 [email] => peter@example.com
 )
```

PHP filter_list() 函数

定义和用法

filter_list() 函数返回包含所有得到支持的过滤器的一个数组。

语法

```
filter_list()
```

提示和注释

注释：该函数的结果不是过滤器的 ID，而是过滤器名称。请使用 filter_id() 函数来获取过滤器 ID。

例子

```
<?php
print_r(filter_list());
?>
```

输出类似：

```
Array
 (
 [0] => int
 [1] => boolean
 [2] => float
 [3] => validate_regexp
 [4] => validate_url
 [5] => validate_email
 [6] => validate_ip
 [7] => string
 [8] => stripped
 [9] => encoded
 [10] => special_chars
 [11] => unsafe_raw
 [12] => email
 [13] => url
 [14] => number_int
 [15] => number_float
 [16] => magic_quotes
 [17] => callback
 )
```

PHP filter_var_array() 函数

定义和用法

filter_var_array() 函数获取多项变量，并进行过滤。

由于无需重复调用 filter_input()，因此本函数对过滤多个变量很有用。

如果成功，则返回包含被过滤的变量值的数组，如果失败，则返回 false。

语法

```
filter_var_array(array, args)
```

参数	描述
array	必需。规定带有字符串键的数组，包含要过滤的数据。
args	可选。规定过滤器参数数组。合法的数组键是变量名。合法的值是过滤器 ID，或者规定过滤器、标志以及选项的数组。该参数也可以是一个单独的过滤器 ID，如果是这样，输入数组中的所有值由指定过滤器进行过滤。

提示和注释

提示：参见完整的 PHP Filter 参考手册，查看可与该函数一同使用的过滤器。

例子

```
<?php
$arr = array
 (
 "name" => "peter griffin",
 "age" => "41",
 "email" => "peter@example.com",
 );

$filters = array
 (
 "name" => array
  (
  "filter"=>FILTER_CALLBACK,
  "flags"=>FILTER_FORCE_ARRAY,
  "options"=>"ucwords"
  ),
 "age" => array
  (
  "filter"=>FILTER_VALIDATE_INT,
  "options"=>array
   (
   "min_range"=>1,
   "max_range"=>120
   )
  ),
 "email"=> FILTER_VALIDATE_EMAIL,
 );

print_r(filter_var_array($arr, $filters));
?>
```

输出类似：

```
Array
 (
 [name] => Peter Griffin
 [age] => 41
 [email] => peter@example.com
 )
```

PHP filter_var() 函数

定义和用法

filter_var() 函数通过指定的过滤器过滤变量。

如果成功，则返回已过滤的数据，如果失败，则返回 false。

语法

```
filter_var(variable, filter, options)
```

参数	描述
variable	必需。规定要过滤的变量。
filter	可选。规定要使用的过滤器的 ID。
options	规定包含标志/选项的数组。检查每个过滤器可能的标志和选项。

提示和注释

提示：参见完整的 PHP Filter 参考手册，查看可与该函数一同使用的过滤器。

例子

```
<?php
if(!filter_var("someone@example....com", FILTER_VALIDATE_EMAIL))
 {
 echo("E-mail is not valid");
 }
else
 {
 echo("E-mail is valid");
 }
?>
```

输出类似：

```
E-mail is not valid
```

PHP FTP 函数

PHP FTP 简介

FTP 函数通过文件传输协议 (FTP) 提供对文件服务器的客户端访问。

FTP 函数用于打开、登录以及关闭连接，同时用于上传、下载、重名命、删除及获取文件服务器上的文件信息。不是所有 FTP 函数对每个服务器都起作用或返回相同的结果。自 PHP 3 起，FTP 函数可用。

这些函数用于对 FTP 服务器进行细致的访问。如果您仅仅需要对 FTP 服务器进行读写操作，建议使用 Filesystem 函数中的 ftp:// wrapper。

安装

PHP 的 Windows 版本已经内置该 FTP 扩展模块的支持。无需加载任何附加扩展库即可使用这些函数。

不过，如果您运行的是 PHP 的 Linux 版本，在编译的时候请添加 --enable-ftp 选项 (PHP4 或以上版本) 或者 --with-ftp (PHP3 版本)。

PHP FTP 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
ftp_alloc()	为要上传到 FTP 服务器的文件分配空间。	5
ftp_cdup()	把当前目录改变为 FTP 服务器上的父目录。	3
ftp_chdir()	改变 FTP 服务器上的当前目录。	3
ftp_chmod()	通过 FTP 设置文件上的权限。	5
ftp_close()	关闭 FTP 连接。	4
ftp_connect()	打开 FTP 连接。	3
ftp_delete()	删除 FTP 服务器上的文件。	3
ftp_exec()	在 FTP 上执行一个程序/命令。	4
ftp_fget()	从 FTP 服务器上下载一个文件并保存到本地一个已经打开的文件中。	3

ftp_fput()	上传一个已打开的文件，并在 FTP 服务器上把它保存为一个文件。	3
ftp_get_option()	返回当前 FTP 连接的各种不同的选项设置。	4
ftp_get()	从 FTP 服务器下载文件。	3
ftp_login()	登录 FTP 服务器。	3
ftp_mdtm()	返回指定文件的最后修改时间。	3
ftp_mkdir()	在 FTP 服务器创建一个新目录。	3
ftp_nb_continue()	连续获取／发送文件 (non-blocking)。	4
ftp_nb_fget()	从FTP服务器上下载文件并保存到本地已经打开的文件中 (non-blocking)	4
ftp_nb_fput()	上传已打开的文件，并在FTP服务器上把它保存为文件 (non-blocking)。	4
ftp_nb_get()	从 FTP 服务器下载文件 (non-blocking)。	4
ftp_nb_put()	把文件上传到服务器 (non-blocking)。	4
ftp_nlist()	返回指定目录的文件列表。	3
ftp_pasv()	返回当前 FTP 被动模式是否打开。	3
ftp_put()	把文件上传到服务器。	3
ftp_pwd()	返回当前目录名称。	3
ftp_quit()	ftp_close() 的别名。	3
ftp_raw()	向 FTP 服务器发送一个 raw 命令。	5
ftp_rawlist()	返回指定目录中文件的详细列表。	3
ftp_rename()	重命名 FTP 服务器上的文件或目录。	3
ftp_rmdir()	删除 FTP 服务器上的目录。	3
ftp_set_option()	设置各种 FTP 运行时选项。	4
ftp_site()	向服务器发送 SITE 命令。	3
ftp_size()	返回指定文件的大小。	3
ftp_ssl_connect()	打开一个安全的 SSL-FTP 连接。	4
ftp_systype()	返回远程 FTP 服务器的系统类型标识符。	3

PHP FTP 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
FTP_ASCII		3
FTP_TEXT		3
FTP_BINARY		3
FTP_IMAGE		3
FTP_TIMEOUT_SEC		3
FTP_AUTOSEEK		4
FTP_AUTORESUME	为 GET 和 PUT 请求自动决定恢复和开始的位置 只能工作在 FTP_AUTOSEEK 打开的情况下	4
FTP_FAILED	异步传输失败	4
FTP_FINISHED	异步传输成功	4
FTP_MOREDATA	异步传输是活动状态的	4

PHP ftp_alloc() 函数

定义和用法

ftp_alloc() 函数为要上传到 FTP 服务器的文件分配空间。

若成功，则返回 true。否则返回 false。

语法

```
ftp_alloc(ftp_connection,size,return)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接。
size	可选。规定要分配的字节数。
return	可选。规定存储服务器响应的字节数。

提示和注释

注释：很多服务器不支持该命令。

例子

例子 1

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_alloc($conn,"160",$response);
echo $response;

ftp_close($conn);
?>
```

例子 2

```
<?php

$file = "myfile.txt";

$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

if (ftp_alloc($conn, filesize($file), $response))
  {
  echo "Space allocated on server.";
  }
else
  {
  echo "Unable to allocate space. " . $response;
  }

ftp_close($conn);

?>
```

PHP ftp_cdup() 函数

定义和用法

ftp_cdup() 函数把当前目录改变为 FTP 服务器上的父目录。

若成功，则返回 true。否则返回 false。

语法

```
ftp_cdup(ftp_connection)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

//输出当前目录
echo "Dir: ".ftp_pwd($conn);
echo "<br />";

//更改为 images 目录
ftp_chdir($conn,"images");
echo "Dir: ".ftp_pwd($conn);
echo "<br />";

//把当前目录切换为父目录
ftp_cdup($conn);
echo "Dir: ".ftp_pwd($conn);

ftp_close($ftp_server);
?>
```

输出：

```
Dir: /
Dir: /images
Dir: /
```

PHP ftp_chdir() 函数

定义和用法

ftp_chdir() 函数改变 FTP 服务器上的当前目录。

若成功，则返回 true。否则返回 false。如果切换目录失败，PHP 还会发出一条警告。

语法

```
ftp_chdir(ftp_connection,directory)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
directory	必需。规定要切换到的目录。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

//输出当前目录
echo "Dir: ".ftp_pwd($conn);
echo "<br />";

//切换为 images 目录
ftp_chdir($conn,"images");
echo "Dir: ".ftp_pwd($conn);

ftp_close($ftp_server);
?>
```

输出：

```
Dir: /
Dir: /images
```

PHP ftp_chmod() 函数

定义和用法

ftp_chmod() 函数设置 FTP 服务器上指定文件的权限。

若成功，则该函数返回新的权限。否则返回 false。

语法

```
ftp_chmod(ftp_connection,mode,file)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
mode	必需。规定新的权限。
file	必需。规定要修改权限的文件的名称。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"user","pass");

// 所有者可读写，其他人没有任何权限
ftp_chmod($conn,"0600","test.txt");

// 所有者可读写，其他人可读
ftp_chmod($conn,"0644","test.txt");

// 所有者拥有所有权限，其他人可读可执行
ftp_chmod($conn,"0755","test.txt");

// 所有者拥有所有权限，所有者所属的组可读
ftp_chmod($conn,"0740","test.txt");

ftp_close($conn);
?>
```

PHP ftp_close() 函数

定义和用法

ftp_close() 函数关闭 FTP 连接。

该函数关闭给出的连接标识符并释放资源。

语法

```
ftp_close(ftp_connection)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");

//要执行的一些代码

ftp_close($conn);
?>
```

PHP ftp_connect() 函数

定义和用法

ftp_connect() 函数建立一个新的 FTP 连接。

若成功，则返回一个连接标识，否则返回 false。

语法

```
ftp_connect(host,port,timeout)
```

参数	描述
host	必需。规定要连接的 FTP 服务器。可以是域名或 IP 地址。后面不应以斜线结尾，前面也不需要用 ftp:// 开头。
port	可选。规定 FTP 服务器的端口。
timeout	可选。规定该 FTP 服务器的超时时间。默认是 90 秒。

说明

提示：超时时间可以在任何时候通过函数 ftp_set_option() 及 ftp_get_option() 来改变和获取。

参数 *timeout* 仅适用于 PHP 4.2.0 及以上版本。

例子

本例尝试连接一个 FTP 服务器。如果连接失败，则 die() 函数将终止脚本，并输出一条消息：

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
?>
```

PHP ftp_delete() 函数

定义和用法

ftp_delete() 函数删除 FTP 服务器上的一个文件。

若成功，则返回 true，否则返回 false。

语法

```
ftp_delete(ftp_connection,path)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
path	必需。规定要删除的文件的路径。

说明

ftp*delete()* 函数用来删除 *FTP* 服务器上的一个由参数 *_path* 指定的的文件。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

echo ftp_delete($conn,"test.txt");

ftp_close($conn);
?>
```

输出：

```
1
```

PHP ftp_exec() 函数

定义和用法

ftp_exec() 函数请求在 FTP 服务器上执行一个程序或命令。

若成功（服务器发送响应代码 200），则返回 true，否则返回 false。

语法

```
ftp_exec(ftp_connection,command)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
command	必需。规定发送到服务器的命名请求。

说明

该函数发送一个 SITE EXEC command 请求到 FTP 服务器。

提示和注释

与 ftp_raw() 函数 不同，ftp_exec() 只有在登录到 FTP 服务器后才能发送命令。

例子

```
<?php
$command = "ls-al > test.txt";
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

if (ftp_exec($conn,$command))
  {
  echo "Command executed successfully";
  }
else
  {
  echo "Execution of command failed";
  }

ftp_close($conn);
?>
```

PHP ftp_fget() 函数

定义和用法

ftp_fget() 函数从 FTP 服务器上下载一个文件并保存到本地一个已经打开的文件中。

若成功则返回 true，失败则返回 false。

语法

```
ftp_fget(ftp_connection,local,remote,mode,resume)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
local	必需。本地已经打开的文件的句柄。
remote	必需。规定从中进行拷贝的文件的路径。
mode	必需。规定传输模式。可能的值有： `FTP_ASCII` `FTP_BINARY`
resume	必需。规定在远程文件中的何处开始拷贝。默认是 0。

说明

参数 *resume* 仅适用于 PHP 4.3.0 以上版本

例子

本例把文本从 "source.txt" 拷贝到 "target.txt" 中：

```
<?php
$source = "source.txt";
$target = fopen("target.txt", "w");

$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_fget($conn,$target,$source,FTP_ASCII);

ftp_close($conn);
?>
```

PHP ftp_fput() 函数

定义和用法

ftp_fput() 函数上传一个已经打开的文件到 FTP 服务器。

若成功则返回 true，失败则返回 false。

语法

```
ftp_fput(ftp_connection,remote,local,mode,resume)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
remote	必需。上传到服务器上的文件名。
local	必需。规定所打开文件的句柄。
mode	必需。规定传输模式。可能的值有： FTP_ASCII FTP_BINARY
resume	必需。规定在本地文件中的何处开始拷贝。默认是 0。

说明

参数 *resume* 仅适用于 PHP 4.3.0 以上版本

例子

本例把文本从 "source.txt" 拷贝到 "target.txt" 中：

```
<?php
$source = fopen("source.txt","r");

$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

echo ftp_fput($conn,"target.txt",$source,FTP_ASCII);

ftp_close($conn);
?>
```

输出：

```
1
```

PHP ftp_get_option() 函数

定义和用法

ftp_get_option() 函数返回当前 FTP 连接的各种不同的选项设置。

语法

```
ftp_get_option(ftp_connection,option)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
option	必需。规定要返回的选项。可能的值有： `FTP_TIMEOUT_SEC` - 返回网络操作的时间限制 `FTP_AUTOSEEK` - 如果设置该选项，则返回 true，否则返回 false

说明

如果成功，则返回选项的值，否则，如果给定的参数 *option* 选项若不被支持，则返回 false 同时会提示一条错误信息。

此函数会返回连接句柄为 *ftp_connection*，指定键值 *option* 的值。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

echo ftp_get_option($conn,FTP_TIMEOUT_SEC);

ftp_close($conn);
?>
```

输出：

```
90
```

PHP ftp_get() 函数

定义和用法

ftp_get() 函数从 FTP 服务器上下载一个文件。

若成功则返回 true，失败则返回 false。

语法

```
ftp_get(ftp_connection,local,remote,mode,resume)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
local	必需。规定本地文件。
remote	必需。规定从中进行拷贝的文件的路径。
mode	必需。规定传输模式。可能的值有： `FTP_ASCII` `FTP_BINARY`
resume	必需。规定在远程文件中的何处开始拷贝。默认是 0。

说明

参数 *resume* 仅适用于 PHP 4.3.0 以上版本

例子

本例把文本从 "source.txt" 拷贝到 "target.txt" 中：

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

echo ftp_get($conn,"target.txt","source.txt",FTP_ASCII);

ftp_close($conn);
?>
```

输出：

```
1
```

PHP ftp_login() 函数

定义和用法

ftp_login() 函数登录 FTP 服务器。

若成功则返回 true，失败则返回 false 并发出一个警告。

语法

```
ftp_login(ftp_connection,username,password)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
username	必需。规定用于登录的用户名。
password	必需。规定用于登录的密码。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");
ftp_close($conn);
?>
```

PHP ftp_mdtm() 函数

定义和用法

ftp_mdtm() 函数返回指定文件的最后修改时间。

语法

```
ftp_login(ftp_connection,file)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
file	必需。规定要检查的文件。

提示和注释

注释：并非所有 FTP 服务器都支持该函数。

注释：该函数不适用于检查目录。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

$mod = ftp_mdtm($conn,"test.txt");

//以 Unix 时间戳返回结果
echo $mod;
echo "<br />";

//把 Unix 时间戳格式化为日期
echo date(DATE_RFC822,$mod);

ftp_close($conn);
?>
```

输出：

```
1140082571
Thu, 16 Feb 2006 10:36:11 CET
```

PHP ftp_mkdir() 函数

定义和用法

ftp_mkdir() 函数在 FTP 服务器上建立新目录。

语法

```
ftp_mkdir(ftp_connection,dir)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
dir	必需。规定要创建的目录的名称。

说明

在 FTP 服务器上建立一个目录名为参数 *dir* 的新目录。

如果成功，则返回新建的目录名，否则返回 false。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

echo ftp_mkdir($conn,"testdir");

ftp_close($conn);
?>
```

输出：

```
/testdir
```

PHP ftp_nb_continue() 函数

定义和用法

ftp_nb_continue() 函数连续获取 / 发送文件。

该函数返回下列值：

- FTP_FAILED (send/receive failed)
- FTP_FINISHED (send/receive completed)
- FTP_MOREDATA (send/receive in progress)

该函数异步地发送/获取文件。这意味着您的程序可以在文件下载时执行其他操作。

语法

```
ftp_nb_continue(ftp_connection)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。

例子

```
<?php
$source = "source.txt";
$target = fopen("target.txt", "w");

$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

$status = ftp_nb_fget($conn,$source,$target,FTP_ASCII);

while ($status == FTP_MOREDATA)
  {
  $status = ftp_nb_continue($conn);
  }

if ($status != FTP_FINISHED)
  {
  echo "Download error";
  }

ftp_close($conn);
?>
```

PHP ftp_nb_fget() 函数

定义和用法

ftp_nb_fget() 函数从 FTP 服务器上下载一个文件并保存到本地已经打开的一个文件中 (non-blocking)。

该函数返回下列值：

- FTP_FAILED (send/receive failed)
- FTP_FINISHED (send/receive completed)
- FTP_MOREDATA (send/receive in progress)

与 ftp_fget() 不同，该函数异步地获取文件。这意味着您的程序可以在文件下载时执行其他操作。

语法

```
ftp_nb_fget(ftp_connection,local,remote,mode,resume)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
local	必需。规定本地文件。
remote	必需。规定从中进行拷贝的文件的路径。
mode	必需。规定传输模式。可能的值有： `FTP_ASCII` `FTP_BINARY`
resume	必需。规定在远程文件中的何处开始拷贝。默认是 0。

例子

本例把文本从 "source.txt" 拷贝到 "target.txt" 中：

```
<?php
$source = "source.txt";
$target = fopen("target.txt", "w");

$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_nb_fget($conn,$target,$source,FTP_ASCII);

ftp_close($conn);
?>
```

PHP ftp_nb_put() 函数

定义和用法

ftp_nb_put() 函数把文件上传到服务器 (non-blocking)。

该函数返回下列值：

- FTP_FAILED (send/receive failed)
- FTP_FINISHED (send/receive completed)
- FTP_MOREDATA (send/receive in progress)

与 ftp_put() 不同，该函数异步地获取文件。这意味着您的程序可以在文件传输时执行其他操作。

语法

```
ftp_nb_fput(ftp_connection,remote,local,mode,resume)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
remote	必需。上传到服务器上的文件名。
local	必需。规定要上传的本地文件的路径。
mode	必需。规定传输模式。可能的值有： FTP_ASCII FTP_BINARY
resume	必需。规定在本地文件中的何处开始拷贝。默认是 0。

例子

本例把文本从 "source.txt" 拷贝到 "target.txt" 中：

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_nb_put($conn,"target.txt","source.txt",FTP_ASCII);

ftp_close($conn);
?>
```

PHP ftp_nlist() 函数

定义和用法

ftp_nlist() 函数返回指定目录的文件列表。

如果成功，则返回给定目录下的文件名组成的数组，否则返回 false。

语法

```
ftp_nlist(ftp_connection,dir)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
dir	必需。规定要检查的目录。使用 "." 来获得当前目录。

提示和注释

注释：该函数不适用于 IIS (Internet Information Server)。它返回 nothing。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

print_r(ftp_nlist($conn,"images"));

ftp_close($conn);
?>
```

输出类似：

```
array(3)
{
[0]=> "flower.gif"
[1]=> "car.gif"
[2]=> "house.gif"
}
```

PHP ftp_pasv() 函数

定义和用法

ftp_pasv() 函数把被动模式设置为打开或关闭。

在被动模式中，数据连接是由客户机来初始化的，而不是服务器。这在客户机位于防火墙之后时比较有用。

语法

```
ftp_pasv(ftp_connection,mode)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
mode	必需。规定模式。 TRUE = passive mode on FALSE = passive mode off

说明

如果参数 *mode* 为真，打开被动模式传输 (PASV MODE) ，否则，如果参数 *mode* 为假，则关闭被动传输模式。在被动模式打开的情况下，数据的传送由客户机启动，而不是由服务器开始。

如果成功则返回 true，失败则返回 false。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_pasv($conn,TRUE);

ftp_close($conn);
?>
```

PHP ftp_put() 函数

定义和用法

ftp_put() 函数把文件上传到服务器。

若成功则返回 true，失败则返回 false。

语法

```
ftp_put(ftp_connection,remote,local,mode,resume)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
remote	必需。上传到服务器上的文件名。
local	必需。规定要上传的本地文件的路径。
mode	必需。规定传输模式。可能的值有： FTP_ASCII FTP_BINARY
resume	必需。规定在本地文件中的何处开始拷贝。默认是 0。

例子

本例把文本从 "source.txt" 拷贝到 "target.txt" 中：

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

echo ftp_put($conn,"target.txt","source.txt",FTP_ASCII);

ftp_close($conn);
?>
```

输出：

```
1
```

PHP ftp_pwd() 函数

定义和用法

ftp_pwd() 函数返回当前目录名。

语法

```
ftp_pwd(ftp_connection)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

// 输出当前目录
echo ftp_pwd($conn) . "<br />";

// 把目录改为 images
ftp_chdir($conn,"images");

// 输出当前目录
echo ftp_pwd($conn);

ftp_close($conn);
?>
```

PHP ftp_quit() 函数

定义和用法

ftp_quit() 函数关闭 FTP 连接。

该函数关闭给出的连接标识符并释放资源。

语法

```
ftp_quit(ftp_connection)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。

提示和注释

提示：该函数是 ftp_close() 函数的别名。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");

//要执行的一些代码

ftp_quit($conn);
?>
```

PHP ftp_raw() 函数

定义和用法

ftp_raw() 函数向 FTP 服务器发送一个 raw 命令。

语法

```
ftp_raw(ftp_connection,command)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
command	必需。规定要执行的命令。

提示和注释

注释：该函数以字符串数组的形式返回服务器的响应。不执行解析，且 ftp_raw() 不检查命令是否正确。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");

print_r (ftp_raw($conn,"USER admin"));
print_r (ftp_raw($conn,"PASS ert456"));

ftp_close($conn);
?>
```

输出：

```
Array ([0] => 331 User admin, password please)
Array ([0] => 230 Password Ok, User logged in)
```

PHP ftp_rawlist() 函数

定义和用法

ftp_rawlist() 函数返回指定目录中文件的详细列表。

语法

```
ftp_rawlist(ftp_connection,dir,recursive)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
dir	必需。规定目录。使用 "." 来规定当前目录。
recursive	可选。默认地，该函数向服务器发送 "LIST" 命令。如果，如果 recursive 参数设置为 true，则发送 "LIST -R" 命令。

说明

ftp_rawlist() 函数将执行 FTP LIST 命令，并把结果返回为一个数组。数组的每个元素为返回文本的每一行，输出结构不会被解析。

使用函数 ftp_systype() 可以用来判断 FTP 服务器的类型，从而可以用来判断返回列表的类型。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

print_r (ftp_rawlist($conn,"."));

ftp_close($conn);
?>
```

输出类似：

```
Array
(
[0] => dr--r--r-- 1 user group 0 Feb 15 13:02 .
[1] => dr--r--r-- 1 user group 0 Feb 15 13:02 ..
[2] => drw-rw-rw- 1 user group 0 Jan 03 08:33 images
[3] => -rw-rw-rw- 1 user group 160 Feb 16 13:54 test.php
[4] => -rw-rw-rw- 1 user group 20 Feb 14 12:22 test.txt
)
```

PHP ftp_rename() 函数

定义和用法

ftp_rename() 函数更改 FTP 服务器上的文件或目录名。

如果成功，则返回 true，否则返回 false。

语法

```
ftp_rename(ftp_connection,from,to)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
from	必需。规定要改名的文件或目录。
to	必需。规定文件或目录的新名称。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_rename($conn,"oldname.txt","newname.txt");

ftp_close($conn);
?>
```

PHP ftp_rmdir() 函数

定义和用法

ftp_rmdir() 函数删除一个目录。

如果成功，则返回 true，否则返回 false。

语法

```
ftp_rmdir(ftp_connection,dir)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
dir	必需。规定要删除的目录。

说明

删除由参数 *dir* 指定的目录。*dir* 必须是一个空目录的绝对或相对路径。

如果成功，则返回 true，否则返回 false。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_rmdir($conn,"testdir");

ftp_close($conn);
?>
```

PHP ftp_set_option() 函数

定义和用法

ftp_set_option() 函数设置各种 FTP 运行时选项。

语法

```
ftp_set_option(ftp_connection,option,value)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
option	必需。规定要删设置的运行时选项。可能的值： `FTP_TIMEOUT_SEC` `FTP_AUTOSEEK` 详细信息见下面的说明。
value	必需。设置 option 参数的值。

说明

FTP_TIMEOUT_SEC 选项改变网络传输的超时时间。参数 value 必须为整数且大于 0。默认的超时时间为 90 秒。

当 FTP_AUTOSEEK 选项打开时，带 resumepos 或 startpos 参数的 GET 或 PUT 请求将先检索到文件中指定的位置。此选项默认是打开的。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_set_option($conn,FTP_TIMEOUT_SEC,120);

ftp_close($conn);
?>
```

PHP ftp_site() 函数

定义和用法

ftp_site() 函数向服务器发送 SITE 命令。

SITE 命令没有标准化。不同的服务器不尽相同。对于处理文件权限或组关系方面的事情，SITE 命令很有用。

语法

```
ftp_site(ftp_connection,command)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
command	必需。规定向 FTP 发送的 SITE 命令。

说明

ftp*site() 函数向 FTP 服务器发送由参数 _command* 指定的命令。

如果成功则返回 true，失败则返回 false。

提示和注释

提示：如需查看哪些命令可用，请通过 ftp_raw() 函数发送 REMOTEHELP 命令。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

ftp_site($conn,"CHMOD 0600 sitetest.txt");

ftp_close($conn);
?>
```

PHP ftp_size() 函数

定义和用法

ftp_size() 函数返回指定文件的大小。

语法

```
ftp_size(ftp_connection,remote_file)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。
remote_file	必需。规定要检查的文件。

说明

ftp*size()* 函数以字节返回远程文件 _*remote_file* 的大小。如果指定文件不存在或发生错误，则返回 -1。有些 FTP 服务器可能不支持此特性。

如果获取成功，则返回文件大小，否则返回 -1。

提示和注释

注释：并非所有 FTP 服务器均支持该函数。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

echo ftp_size($conn,"test.txt");

ftp_close($conn);
?>
```

输出类似：

```
160
```

PHP ftp_ssl_connect() 函数

定义和用法

ftp_ssl_connect() 函数打开一个安全的 SSL-FTP 连接。

当连接打开，您就可以在服务器运行 FTP 函数。

语法

```
ftp_ssl_connect(host,port,timeout)
```

参数	描述
host	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。可以是域名或 IP 地址。该参数不能包含 "ftp://" 或 斜杠。
port	可选。规定 FTP 服务器的端口。默认是 21。
timeout	可选。规定 FTP 连接的超时时间。默认是 90 秒。

例子

本例尝试连接一个 FTP 服务器。如果连接失败，die() 函数会终止脚本的执行，并输出一条消息：

```
<?php
$conn = ftp_ssl_connect("ftp.testftp.com") or die("Could not connect");
?>
```

PHP ftp_systype() 函数

定义和用法

ftp_systype() 函数返回远程 FTP 服务器的系统类型标识符。

该函数返回远程服务器的系统类型。若发生错误，则返回 false。

语法

```
ftp_systype(ftp_connection)
```

参数	描述
ftp_connection	必需。规定要使用的 FTP 连接（FTP 连接的标识符）。

例子

```
<?php
$conn = ftp_connect("ftp.testftp.com") or die("Could not connect");
ftp_login($conn,"admin","ert456");

echo ftp_systype($conn);

ftp_close($conn);
?>
```

输出类似：

```
UNIX
```

PHP HTTP 函数

PHP HTTP 简介

HTTP 函数允许您在其他输出被发送之前，对由 web 服务器发送到浏览器的信息进行操作。

安装

HTTP 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP HTTP 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
header()	向客户端发送原始的 HTTP 报头。	3
headers_list()	返回已发送的（或待发送的）响应头部的一个列表。	5
headers_sent()	检查 HTTP 报头是否发送/已发送到何处。	3
setcookie()	向客户端发送一个 HTTP cookie。	3
setrawcookie()	不对 cookie 值进行 URL 编码，发送一个 HTTP cookie。	5

PHP header() 函数

定义和用法

header() 函数向客户端发送原始的 HTTP 报头。

认识到一点很重要，即必须在任何实际的输出被发送之前调用 header() 函数（在 PHP 4 以及更高的版本中，您可以使用输出缓存来解决此问题）：

```
<html>
<?php
// 结果出错
// 在调用 header() 之前已存在输出
header('Location: http://www.example.com/');
?>
```

语法

```
header(string,replace,http_response_code)
```

参数	描述
string	必需。规定要发送的报头字符串。
replace	可选。指示该报头是否替换之前的报头，或添加第二个报头。默认是 true（替换）。false（允许相同类型的多个报头）。
http_response_code	可选。把 HTTP 响应代码强制为指定的值。（PHP 4 以及更高版本可用）

提示和注释

注释：从 PHP 4.4 之后，该函数防止一次发送多个报头。这是对头部注入攻击的保护措施。

例子

例子 1

```
<?php
// Date in the past
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Cache-Control: no-cache");
header("Pragma: no-cache");
?>

<html>
<body>

...
...
```

注释：用户可能会设置一些选项来更改浏览器的默认缓存设置。通过发送上面的报头，您可以覆盖任何这些设置，强制浏览器不进行缓存！

例子 2

提示用户保存一个生成的 PDF 文件（Content-Disposition 报头用于提供一个推荐的文件名，并强制浏览器显示保存对话框）：

```
<?php
header("Content-type:application/pdf");

// 文件将被称为 downloaded.pdf
header("Content-Disposition:attachment;filename='downloaded.pdf'");

// PDF 源在 original.pdf 中
readfile("original.pdf");
?>

<html>
<body>

...
...
```

注释：微软 IE 5.5 存在一个阻止以上机制的 bug。通过升级为 Service Pack 2 或更高的版本，可以解决该 bug。

PHP headers_list() 函数

定义和用法

headers_list() 函数返回已发送的（或待发送的）响应头部的一个列表。

该函数返回包含报头的数组。

语法

```
headers_list()
```

提示和注释

提示：如需确定是否已发送报头，请使用 headers_sent() 函数。

例子

```
<?php
setcookie("TestCookie","SomeValue");
header("X-Sample-Test: foo");
header('Content-type: text/plain');
?>

<html>
<body>

<?php
// 发送哪些报头？
var_dump(headers_list());
?>

</body>
</html>
```

PHP headers_sent() 函数

定义和用法

headers_sent() 函数检查 HTTP 标头是否已被发送以及在哪里被发送。

如果报头已发送，则返回 true，否则返回 false。

语法

```
headers_sent(_file_,_line_)
```

参数	描述
file,line	可选。如果设置 *file* 和 *line* 参数，headers_sent() 会把输出开始的 PHP 源文件名和行号存入 file 和 line 变量中。

提示和注释

注释：一旦报头块已经发送，就不能使用 header() 函数 来发送其它的标头。使用此函数至少可以避免与 HTTP 标头有关的错误信息。

注释：可选的 *file* 和 *line* 参数是 PHP 4.3 中新加的。

例子

例子 1

```
<?php
// 如果报头未发送，则发送一个
if (!headers_sent())
  {
  header("Location: http://www.w3school.com.cn/");
  exit;
  }
?>

<html>
<body>

...
...
```

例子 2

使用可选的 file 和 line 参数：

```
<?php
// 传递 $file 和 $line，供日后使用
// 不要预先为它们赋值
if (!headers_sent($file, $line))
  {
  header("Location: http://www.w3school.com.cn/");
  exit;
  // Trigger an error here
  }
else
  {
  echo "Headers sent in $file on line $line";
  exit;
  }
?>

<html>
<body>

...
...
```

PHP setcookie() 函数

定义和用法

setcookie() 函数向客户端发送一个 HTTP cookie。

cookie 是由服务器发送到浏览器的变量。cookie 通常是服务器嵌入到用户计算机中的小文本文件。每当计算机通过浏览器请求一个页面，就会发送这个 cookie。

cookie 的名称指定为相同名称的变量。例如，如果被发送的 cookie 名为 "name"，会自动创建名为 $user 的变量，包含 cookie 的值。

必须在任何其他输出发送前对 cookie 进行赋值。

如果成功，则该函数返回 true，否则返回 false。

语法

```
setcookie(name,value,expire,path,domain,secure)
```

参数	描述
name	必需。规定 cookie 的名称。
value	必需。规定 cookie 的值。
expire	可选。规定 cookie 的有效期。
path	可选。规定 cookie 的服务器路径。
domain	可选。规定 cookie 的域名。
secure	可选。规定是否通过安全的 HTTPS 连接来传输 cookie。

提示和注释

注释：可以通过 $HTTP_COOKIE_VARS["user"] 或 $_COOKIE["user"] 来访问名为 "user" 的 cookie 的值。

注释：在发送 cookie 时，cookie 的值会自动进行 URL 编码。接收时会进行 URL 解码。如果你不需要这样，可以使用 setrawcookie() 代替。

例子

例子 1

设置并发送 cookie：

```
<?php
$value = "my cookie value";

// 发送一个简单的 cookie
setcookie("TestCookie",$value);
?>

<html>
<body>

...
...
```

```
<?php
$value = "my cookie value";

// 发送一个 24 小时候过期的 cookie
setcookie("TestCookie",$value, time()+3600*24);
?>

<html>
<body>

...
...
```

例子 2

检索 cookie 值的不同方法：

```
<html>
<body>

<?php

// 输出个别的 cookie
echo $_COOKIE["TestCookie"];
echo "<br />";
echo $HTTP_COOKIE_VARS["TestCookie"];
echo "<br />";

// 输出所有 cookie
print_r($_COOKIE);
?>

</body>
</html>
```

输出：

```
my cookie value
my cookie value
Array ([TestCookie] => my cookie value)
```

例子 3

通过把失效日期设置为过去的日期/时间，删除一个 cookie：

```
<?php
// 把失效日期设置为一小时前
setcookie ("TestCookie", "", time() - 3600);
?>

<html>
<body>

...
...
```

例子 4

创建一个数组 cookie：

```
<?php
setcookie("cookie[three]","cookiethree");
setcookie("cookie[two]","cookietwo");
setcookie("cookie[one]","cookieone");

// 输出 cookie （在重载页面后）
if (isset($_COOKIE["cookie"]))
  {
  foreach ($_COOKIE["cookie"] as $name => $value)
    {
    echo "$name : $value <br />";
    }
  }
?>

<html>
<body>

...
...
```

输出：

```
three : cookiethree
two : cookietwo
one : cookieone
```

PHP setrawcookie() 函数

定义和用法

setrawcookie() 函数不对 cookie 值进行 URL 编码，发送一个 HTTP cookie。

cookie 是由服务器发送到浏览器的变量。cookie 通常是服务器嵌入到用户计算机中的小文本文件。每当计算机通过浏览器请求一个页面，就会发送这个 cookie。

cookie 的名称指定为相同名称的变量。例如，如果被发送的 cookie 名为 "name"，会自动创建名为 $user 的变量，包含 cookie 的值。

必须在任何其他输出发送前对 cookie 进行赋值。

如果成功，则该函数返回 true，否则返回 false。

语法

```
setcookie(name,value,expire,path,domain,secure)
```

参数	描述
name	必需。规定 cookie 的名称。
value	必需。规定 cookie 的值。
expire	可选。规定 cookie 的有效期。
path	可选。规定 cookie 的服务器路径。
domain	可选。规定 cookie 的域名。
secure	可选。规定是否通过安全的 HTTPS 连接来传输 cookie。

提示和注释

注释：可以通过 $HTTP_COOKIE_VARS["user"] 或 $_COOKIE["user"] 来访问名为 "user" 的 cookie 的值。

注释：setrawcookie() 与 setcookie() 几乎完全相同，不同的是不会在发往客户机时，对 cookie 值进行自动 URL 编码。

例子

例子 1

设置并发送 cookie：

```
<?php
$value = "my cookie value";

// 发送一个简单的 cookie
setrawcookie("TestCookie",$value);
?>

<html>
<body>

...
...
```

```
<?php
$value = "my cookie value";

// 发送一个 24 小时候过期的 cookie
setrawcookie("TestCookie",$value, time()+3600*24);
?>

<html>
<body>

...
...
```

例子 2

检索 cookie 值的不同方法：

```
<html>
<body>

<?php

// 输出个别的 cookie
echo $_COOKIE["TestCookie"];
echo "<br />";
echo $HTTP_COOKIE_VARS["TestCookie"];
echo "<br />";

// 输出所有 cookie
print_r($_COOKIE);
?>

</body>
</html>
```

输出：

```
my cookie value
my cookie value
Array ([TestCookie] => my cookie value)
```

例子 3

通过把失效日期设置为过去的日期/时间，删除一个 cookie：

```
<?php
// 把失效日期设置为一小时前
setrawcookie ("TestCookie", "", time() - 3600);
?>

<html>
<body>

...
...
```

例子 4

创建一个数组 cookie：

```
<?php
setrawcookie("cookie[three]","cookiethree");
setrawcookie("cookie[two]","cookietwo");
setrawcookie("cookie[one]","cookieone");

// 输出 cookie （在重载页面后）
if (isset($_COOKIE["cookie"]))
  {
  foreach ($_COOKIE["cookie"] as $name => $value)
    {
    echo "$name : $value <br />";
    }
  }
?>

<html>
<body>

...
...
```

输出：

```
three : cookiethree
two : cookietwo
one : cookieone
```

PHP libxml 函数

PHP libxml 简介

libxml 函数和常量与 SimpleXML, XSLT 以及 DOM 一起使用。

安装

这些函数需要 libxml 程序包。在 xmlsoft.org 下载。

PHP libxml 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
libxml_clear_errors()	清空 libxml 错误缓冲。	5
libxml_get_errors()	检索错误数组。	5
libxml_get_last_error()	从 libxml 检索最后的错误。	5
libxml_set_streams_context()	为下一次 libxml 文档加载或写入设置流环境。	5
libxml_use_internal_errors()	禁用 libxml 错误，允许用户按需读取错误信息。	5

PHP libxml 常量

函数	描述	PHP
LIBXML_COMPACT	设置小型节点分配优化。会改善应用程序的性能。	5
LIBXML_DTDATTR	设置默认 DTD 属性。	5
LIBXML_DTDLOAD	加载外部子集。	5
LIBXML_DTDVALID	通过 DTD 进行验证。	5
LIBXML_NOBLANKS	删除空节点。	5
LIBXML_NOCDATA	把 CDATA 设置为文本节点。	5
LIBXML_NOEMPTYTAG	更改空标签（比如 改为 </br>）。仅在 DOMDocument->save() 和 DOMDocument->saveXML() 函数中可用。	5
LIBXML_NOENT	替代实体。	5
LIBXML_NOERROR	不显示错误报告。	5
LIBXML_NONET	在加载文档时停止网络访问。	5
LIBXML_NOWARNING	不显示警告报告。	5
LIBXML_NOXMLDECL	在保存文档时，撤销 XML 声明。	5
LIBXML_NSCLEAN	删除额外的命名空间声明。	5
LIBXML_XINCLUDE	使用 XInclude 置换。	5
LIBXML_ERR_ERROR	获得可恢复的错误。	5
LIBXML_ERR_FATAL	获得致命错误。	5
LIBXML_ERR_NONE	获得无错误。	5
LIBXML_ERR_WARNING	获得简单警告。	5
LIBXML_VERSION	获得 libxml 版本（例如：20605 或 20617）。	5
LIBXML_DOTTED_VERSION	获得有点号的 libxml 版本（例如：2.6.5 或 2.6.17）。	5

PHP libxml_clear_errors() 函数

定义和用法

libxml_clear_errors() 函数清空 libxml 错误缓冲。

语法

```
libxml_clear_errors()
```

例子

```
<?php
libxml_clear_errors()
?>
```

PHP libxml_get_errors() 函数

定义和用法

libxml_get_errors() 函数从 libxml 错误缓冲中获取错误。

该函数返回错误对象的一个数组，如果 libxml 错误缓冲中没有错误，则返回一个空数组。

语法

```
libxml_get_errors()
```

例子

```
<?php
libxml_get_errors()
?>
```

PHP libxml_get_last_error() 函数

定义和用法

libxml_get_last_error() 函数从 libxml 错误缓冲中获取最后一个错误。

若成功，则返回一个错误对象。若失败，或 libxml 错误缓冲中没有错误，则返回 false。

语法

```
libxml_get_last_error()
```

例子

```
<?php
libxml_get_last_error()
?>
```

PHP libxml_use_internal_errors() 函数

定义和用法

libxml_use_internal_errors() 函数禁用标准的 libxml 错误，并启用用户错误处理。

语法

```
libxml_use_internal_errors(user_errors)
```

参数	描述
user_errors	可选。规定是否应该启用用户错误处理。默认是 false。

说明

该函数返回 *user_errors* 参数之前的值。

例子

```
<?php
libxml_use_internal_errors()
?>
```

PHP Mail 函数

PHP Mail 简介

HTTP 函数允许您从脚本中直接发送电子邮件。

需求

要使邮件函数可用，PHP 需要已安装且正在运行的邮件系统。要使用的程序是由 php.ini 文件中的配置设置定义的。

安装

邮件函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

运行时配置

邮件函数的行为受 php.ini 的影响。

Mail 配置选项

名称	默认	描述	可更改
SMTP	"localhost"	Windows 专用：SMTP 服务器的 DNS 名称或 IP 地址。	PHP_INI_ALL
smtp_port	"25"	Windows 专用：SMTP 端口号。自 PHP 4.3 起可用。	PHP_INI_ALL
sendmail_from	NULL	Windows 专用：规定从 PHP 发送的邮件中使用的 "from" 地址。	PHP_INI_ALL
sendmail_path	NULL	Unix 系统专用：规定sendmail 程序的路径（通常 /usr/sbin/sendmail 或 /usr/lib/sendmail）	PHP_INI_SYSTEM

PHP Mail 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
ezmlm_hash()	计算 EZMLM 邮件列表系统所需的散列值。	3
mail()	允许您从脚本中直接发送电子邮件。	3

PHP mail() 函数

定义和用法

mail() 函数允许您从脚本中直接发送电子邮件。

如果邮件的投递被成功地接收，则返回 true，否则返回 false。

语法

```
mail(to,subject,message,headers,parameters)
```

参数	描述
to	必需。规定邮件的接收者。
subject	必需。规定邮件的主题。该参数不能包含任何换行字符。
message	必需。规定要发送的消息。
headers	必需。规定额外的报头，比如 From, Cc 以及 Bcc。
parameters	必需。规定 sendmail 程序的额外参数。

说明

在 *message* 参数规定的消息中，行之间必须以一个 LF（\n）分隔。每行不能超过 70 个字符。

（Windows 下）当 PHP 直接连接到 SMTP 服务器时，如果在一行开头发现一个句号，则会被删掉。要避免此问题，将单个句号替换成两个句号。

```
<?php
$text = str_replace("\n.", "\n..", $text);
?>
```

提示和注释

注释：您需要紧记，邮件投递被接受，并不意味着邮件到达了计划的目的地。

例子

例子 1

发送一封简单的邮件：

```
<?php

$txt = "First line of text\nSecond line of text";

// 如果一行大于 70 个字符，请使用 wordwrap()。
$txt = wordwrap($txt,70);

// 发送邮件
mail("somebody@example.com","My subject",$txt);
?>
```

例子 2

发送带有额外报头的 email：

```
<?php

$to = "somebody@example.com";
$subject = "My subject";
$txt = "Hello world!";
$headers = "From: webmaster@example.com" . "\r\n" .
"CC: somebodyelse@example.com";

mail($to,$subject,$txt,$headers);
?>
```

例子 3

发送一封 HTML email：

```
<?php

$to = "somebody@example.com, somebodyelse@example.com";
$subject = "HTML email";

$message = "
<html>
<head>
<title>HTML email</title>
</head>
<body>
<p>This email contains HTML Tags!</p>
<table>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>
<tr>
<td>John</td>
<td>Doe</td>
</tr>
</table>
</body>
</html>
";

// 当发送 HTML 电子邮件时，请始终设置 content-type
$headers = "MIME-Version: 1.0" . "\r\n";
$headers .= "Content-type:text/html;charset=iso-8859-1" . "\r\n";

// 更多报头
$headers .= 'From: <webmaster@example.com>' . "\r\n";
$headers .= 'Cc: myboss@example.com' . "\r\n";

mail($to,$subject,$message,$headers);
?>
```

PHP Math 函数

PHP Math 简介

数学 (Math) 函数能处理 integer 和 float 范围内的值。

安装

数学 (Math) 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP Math 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
abs()	绝对值。	3
acos()	反余弦。	3
acosh()	反双曲余弦。	4
asin()	反正弦。	3
asinh()	反双曲正弦。	4
atan()	反正切。	3
atan2()	两个参数的反正切。	3
atanh()	反双曲正切。	4
base_convert()	在任意进制之间转换数字。	3
bindec()	把二进制转换为十进制。	3
ceil()	向上舍入为最接近的整数。	3
cos()	余弦。	3
cosh()	双曲余弦。	4
decbin()	把十进制转换为二进制。	3
dechex()	把十进制转换为十六进制。	3
decoct()	把十进制转换为八进制。	3
deg2rad()	将角度转换为弧度。	3

exp()	返回 E^x 的值。	3
expm1()	返回 E^x - 1 的值。	4
floor()	向下舍入为最接近的整数。	3
fmod()	返回除法的浮点数余数。	4
getrandmax()	显示随机数最大的可能值。	3
hexdec()	把十六进制转换为十进制。	3
hypot()	计算直角三角形的斜边长度。	4
is_finite()	判断是否为有限值。	4
is_infinite()	判断是否为无限值。	4
is_nan()	判断是否为合法数值。	4
lcg_value()	返回范围为 (0, 1) 的一个伪随机数。	4
log()	自然对数。	3
log10()	以 10 为底的对数。	3
log1p()	返回 log(1 + number)。	4
max()	返回最大值。	3
min()	返回最小值。	3
mt_getrandmax()	显示随机数的最大可能值。	3
mt_rand()	使用 Mersenne Twister 算法返回随机整数。	3
mt_srand()	播种 Mersenne Twister 随机数生成器。	3
octdec()	把八进制转换为十进制。	3
pi()	返回圆周率的值。	3
pow()	返回 x 的 y 次方。	3
rad2deg()	把弧度数转换为角度数。	3
rand()	返回随机整数。	3
round()	对浮点数进行四舍五入。	3
sin()	正弦。	3
sinh()	双曲正弦。	4
sqrt()	平方根。	3
srand()	播下随机数发生器种子。	3
tan()	正切。	3
tanh()	双曲正切。	4

PHP Math 常量

常量名	常量名	常量值	PHP
M_E	e	2.7182818284590452354	4
M_EULER	Euler 常量	0.57721566490153286061	5.2.0
M_LNPI	log_e(pi)	1.14472988584940017414	5.2.0
M_LN2	log_e 2	0.69314718055994530942	4
M_LN10	log_e 10	2.30258509299404568402	4
M_LOG2E	log_2 e	1.4426950408889634074	4
M_LOG10E	log_10 e	0.43429448190325182765	4
M_PI	Pi	3.14159265358979323846	3
M_PI_2	pi/2	1.57079632679489661923	4
M_PI_4	pi/4	0.78539816339744830962	4
M_1_PI	1/pi	0.31830988618379067154	4
M_2_PI	2/pi	0.63661977236758134308	4
M_SQRTPI	sqrt(pi)	1.77245385090551602729	5.2.0
M_2_SQRTPI	2/sqrt(pi)	1.12837916709551257390	4
M_SQRT1_2	1/sqrt(2)	0.70710678118654752440	4
M_SQRT2	sqrt(2)	1.41421356237309504880	4
M_SQRT3	sqrt(3)	1.73205080756887729352	5.2.0

PHP abs() 函数

定义和用法

abs() 函数返回一个数的绝对值。

语法

```
abs(x)
```

参数	描述
x	必需。一个数。

说明

返回参数 *x* 的绝对值。如果参数 *x* 是 float，则返回的类型也是 float，否则返回 integer（因为 float 通常比 integer 有更大的取值范围）。

例子

```
<?php
echo(abs(6.7));
echo(abs(-3));
echo(abs(3));
?>
```

输出：

```
6.7
3
3
```

PHP acos() 函数

定义和用法

acos() 函数返回一个数的反余弦。

语法

```
acos(x)
```

参数	描述
x	必需。一个数，范围在 -1 到 1 之间。

说明

返回 *x* 的反余弦值，单位是弧度。acos() 是 cos() 的反函数，它的意思是在 acos() 范围里的每个值都是 a==cos(acos(a)) 。

如果 *x* 的值在 -1 到 1 之外，则返回 -1.#IND。如果为 -1，则返回 PI 的值。

例子

在本例中，我们将计算不同值的反余弦：

```
<?php
echo(acos(0.64));
echo(acos(0));
echo(acos(-1));
echo(acos(1));
echo(acos(2));
?>
```

输出：

```
0.876298061168
1.57079632679
3.14159265359
0
-1.#IND
```

PHP acosh() 函数

定义和用法

acosh() 函数返回一个数的反双曲余弦。

语法

```
acosh(x)
```

参数	描述
x	必需。一个数。

说明

返回 *x* 的反双曲余弦值，即，其双曲余弦为 *x* 的那个值。

提示和注释

注释：本函数未在 Windows 平台下实现。

PHP asin() 函数

定义和用法

asin() 函数返回不同数值的反正弦，返回的结果是介于 -PI/2 与 PI/2 之间的弧度值。

语法

```
asin(x)
```

参数	描述
x	必需。一个数，范围在 -1 到 1 之间。

说明

返回 *x* 的反正弦值，单位是弧度。asin() 是 sin() 的反函数，它的意思是在 asin() 范围里的每个值都是 a==sin(asin(a))。

如果 *x* 的值在 -1 到 1 之外，则返回 -1.#IND。如果为 1，则返回 PI/2 的值。

例子

在本例中，我们将计算不同值的反正弦：

```
<?php
echo(asin(0.64));
echo(asin(0));
echo(asin(-1));
echo(asin(1));
echo(asin(2))
?>
```

输出：

```
0.694498265627
0
-1.57079632679
1.57079632679
-1.#IND
```

PHP asinh() 函数

定义和用法

asinh() 函数返回一个数的反双曲正弦。

语法

```
asinh(x)
```

参数	描述
x	必需。一个数。

说明

返回 *x* 的反双曲正弦值，即，其双曲正弦为 *x* 的那个值。

提示和注释

注释：本函数未在 Windows 平台下实现。

PHP atan() 和 atan2() 函数

定义和用法

atan() 函数返回一个数值的反正切，返回值介于 -PI/2 与 PI/2 之间。

atan2() 函数返回两个参数的反正切，返回值为弧度，其值在 -PI 和 PI 之间（包括 -PI 和 PI）。

语法

```
atan(x)

atan2(x,y)
```

参数	描述
x	必需。一个数。
y	必需。一个数。

说明

atan() 函数返回 *x* 的反正切值，单位是弧度。atan() 是 tan() 的反函数，它的意思是在 atan() 范围里的每个值都是 a==tan(atan(a))。

atan2() 函数计算两个变量 *x* 和 *y* 的反正切值。和计算 y / x 的反正切相似，不同的是两个参数的符号是用来确定结果的象限之外。

例子 1

本例计算不同值的反正切：

```
<?php
echo(atan(0.50));
echo(atan(-0.50));
echo(atan(5));
echo(atan(10));
echo(atan(-5));
echo(atan(-10))
?>
```

输出：

```
0.463647609001
-0.463647609001
1.37340076695
1.4711276743
-1.37340076695
-1.4711276743
```

例子 2

本例计算不同的变量 x 和 y 的反正切值：

```
<?php
echo(atan2(0.50,0.50));
echo(atan2(-0.50,-0.50));
echo(atan2(5,5));
echo(atan2(10,20));
echo(atan2(-5,-5));
echo(atan2(-10,10))
?>
```

输出：

```
0.785398163397
-2.35619449019
0.785398163397
0.463647609001
-2.35619449019
-0.785398163397
```

PHP atanh() 函数

定义和用法

atanh() 函数返回一个角度的反双曲正切。

语法

```
atanh(x)
```

参数	描述
x	必需。一个数。

说明

atanh() 函数返回 *x* 的反双曲正切值，即，其双曲正切为 *x* 的那个值。

提示和注释

注释：本函数未在 Windows 平台下实现。

PHP base_convert() 函数

定义和用法

base_convert() 函数在任意进制之间转换数字。

语法

```
base_convert(number,frombase,tobase)
```

参数	描述
number	必需。原始值。
frombase	必需。数字原来的进制。
tobase	必需。要转换的进制。

说明

返回一个字符串，包含 *number* 以 *tobase* 进制的表示。*number* 本身的进制由 *frombase* 指定。*frombase* 和 *tobase* 都只能在 2 和 36 之间（包括 2 和 36）。高于十进制的数字用字母 a-z 表示，例如 a 表示 10，b 表示 11 以及 z 表示 35。

例子 1

把八进制数转换为十进制数：

```
<?php
$oct = "0031";
$dec = base_convert($oct,8,10);
echo "八进制的 $oct 等于十进制的 $dec。";
?>
```

输出：

```
八进制的 0031 等于十进制的 25。
```

例子 2

把八进制数转换为十六进制数：

```
<?php
$oct = "364";
$hex = base_convert($oct,8,16);
echo "八进制的 $oct 等于十六进制的 $hex。";
?>
```

输出：

```
八进制的 364 等于十六进制的 f4。
```

PHP bindec() 函数

定义和用法

bindec() 函数把二进制转换为十进制。

语法

```
bindec(binary_string)
```

参数	描述
binary_string	必需。规定要转换的二进制数。

说明

返回 *binary_string* 参数所表示的二进制数的十进制等价值。

bindec() 函数将一个二进制数转换成 integer。可转换的最大的数为 31 位 1 或者说十进制的 2147483647。PHP 4.1.0 开始，该函数可以处理大数值，这种情况下，它会返回 float 类型。

例子

```
<?php
echo bindec("0011");
echo bindec("01");
echo bindec("11000110011");
echo bindec("111");
?>
```

输出：

```
3
1
1587
7
```

PHP ceil() 函数

定义和用法

ceil() 函数向上舍入为最接近的整数。

语法

```
ceil(x)
```

参数	描述
x	必需。一个数。

说明

返回不小于 *x* 的下一个整数，*x* 如果有小数部分则进一位。ceil() 返回的类型仍然是 float，因为 float 值的范围通常比 integer 要大。

例子

在本例中，我们将对不同的值应用 ceil() 函数：

```
<?php
echo(ceil(0.60);
echo(ceil(0.40);
echo(ceil(5);
echo(ceil(5.1);
echo(ceil(-5.1);
echo(ceil(-5.9));
?>
```

输出：

```
1
1
5
6
-5
-5
```

PHP cos() 函数

定义和用法

cos() 函数返回一个数的余弦。

语法

```
cos(x)
```

参数	描述
x	必需。一个数。

说明

cos() 返回参数 *x* 的余弦值。参数 *x* 的单位为弧度。

提示和注释

注释：cos() 返回的数值在 -1 和 1 之间。

例子

在本例中，我们将计算不同值的余弦：

```
<?php
echo(cos(3));
echo(cos(-3));
echo(cos(0));
echo(cos(M_PI));
echo(cos(2*M_PI));
?>
```

输出：

```
-0.9899924966004454
-0.9899924966004454
1
-1
1
```

PHP cosh() 函数

定义和用法

cosh() 函数返回一个数的双曲余弦。

语法

```
cosh(x)
```

参数	描述
x	必需。一个数。

说明

返回 *x* 的双曲余弦值，定义为 (exp(arg) + exp(-arg))/2。

例子

在本例中，我们将返回不同数的双曲余弦：

```
<?php
echo(cosh(3);
echo(cosh(-3);
echo(cosh(0);
echo(cosh(M_PI);
echo(cosh(2*M_PI));
?>
```

输出：

```
10.0676619958
10.0676619958
1
11.5919532755
267.746761484
```

PHP decbin() 函数

定义和用法

decbin() 函数把十进制转换为二进制。

语法

```
decbin(dec_number)
```

参数	描述
dec_number	必需。规定要转换的十进制数。

说明

返回一个字符串，包含有给定 *dec_number* 参数的二进制表示。所能转换的最大数值为十进制的 4294967295，其结果为 32 个 1 的字符串。

例子

```
<?php
echo decbin("3");
echo decbin("1");
echo decbin("1587");
echo decbin("7");
?>
```

输出：

```
11
1
11000110011
111
```

PHP dechex() 函数

定义和用法

dechex() 函数把十进制转换为十六进制。

语法

```
dechex(dec_number)
```

参数	描述
dec_number	必需。规定要转换的十进制数。

说明

返回一个字符串，包含有给定 *binary_string* 参数的十六进制表示。所能转换的最大数值为十进制的 4294967295，其结果为 "ffffffff"。

例子

```
<?php
echo dechex("30");
echo dechex("10");
echo dechex("1587");
echo dechex("70");
?>
```

输出：

```
1e
a
633
46
```

PHP decoct() 函数

定义和用法

decoct() 函数把十进制转换为八进制。

语法

```
decoct(dec_number)
```

参数	描述
dec_number	必需。规定要转换的十进制数。

说明

返回一个字符串，包含有给定 *dec_number* 参数的八进制表示。所能转换的最大数值为十进制的 4294967295，其结果为 "37777777777"。

例子

```
<?php
echo decoct("30");
echo decoct("10");
echo decoct("1587");
echo decoct("70");
?>
```

输出：

```
36
12
3063
106
```

PHP deg2rad() 函数

定义和用法

deg2rad() 函数将角度转换为弧度。

语法

```
deg2rad(degree_number)
```

参数	描述
degree_number	必需。规定要转换的角度。

说明

本函数把 *degree_number* 从角度转换成弧度。

例子 1

```
<?php
echo deg2rad("30");
echo deg2rad("10");
echo deg2rad("1587");
echo deg2rad("70");
?>
```

输出：

```
0.523598775598
0.174532925199
27.6983752292
1.2217304764
```

例子 2

```
<?php
$deg = 180;
$rad = deg2rad($deg);
echo "角度 $deg 等于弧度 $rad";
?>
```

输出：

```
角度 180 等于弧度 3.14159265359
```

PHP exp() 函数

定义和用法

exp() 函数计算 e 的指数。

语法

```
exp(x)
```

参数	描述
x	必需。一个数。

说明

返回 e 的 *x* 次方值。。

提示和注释

提示：用 'e' 作为自然对数的底 2.718282。

例子

在本例中，我们将对不同的数应用 exp() 函数：

```
<?php
echo(exp(1));
echo(exp(-1));
echo(exp(5));
echo(exp(10))
?>
```

输出：

```
2.718281828459045
0.36787944117144233
148.4131591025766
22026.465794806718
```

PHP expm1() 函数

定义和用法

expm1() 函数返回 exp(x) - 1，甚至当 number 的值接近零也能计算出准确结果。

语法

```
expm1(x)
```

参数	描述
x	必需。一个数。

说明

expm1() 返回 'exp(x) - 1'，甚至当 *x* 的值接近零也能计算出准确结果。但是当两个数值趋近于相等的时候，'exp (x) - 1' 就会变得不太准确。

提示和注释

警告：本函数是实验性的。本函数的行为，包括函数名称以及其它任何关于本函数的文档可能会在没有通知的情况下随 PHP 以后的发布而改变。使用本函数风险自担。

提示：用 'e' 作为自然对数的底 2.718282。

注释：本函数未在 Windows 平台下实现。

例子

在本例中，我们将对不同的数应用 expm1() 函数：

```
<?php
echo(expm1(1));
echo(expm1(-1));
echo(expm1(5));
echo(expm1(10))
?>
```

PHP floor() 函数

定义和用法

floor() 函数向下舍入为最接近的整数。

语法

```
floor(x)
```

参数	描述
x	必需。一个数。

说明

返回不大于 *x* 的下一个整数，将 *x* 的小数部分舍去取整。floor() 返回的类型仍然是 float，因为 float 值的范围通常比 integer 要大。

例子

在本例中，我们将对不同的数应用 floor() 函数：

```
<?php
echo(floor(0.60));
echo(floor(0.40));
echo(floor(5));
echo(floor(5.1));
echo(floor(-5.1));
echo(floor(-5.9))
?>
```

输出：

```
0
0
5
5
-6
-6
```

PHP fmod() 函数

定义和用法

fmod() 函数返回除法的浮点数余数。

语法

```
fmod(x,y)
```

参数	描述
x	必需。一个数。
y	必需。一个数。

说明

返回被除数（*x*）除以除数（*y*）所得的浮点数余数。余数（r）的定义是：x = i * y + r，其中 *i* 是整数。如果 *y* 是非零值，则 *r* 和 *x* 的符号相同并且其数量值小于 *y* 。

例子

在本例中，我们将使用 fmod() 函数来返回 5/2 的余数：

```
<?php
$r = fmod(5,2);
echo $r
?>
```

输出：

```
1
```

PHP hexdec() 函数

定义和用法

hexdec() 函数把十六进制转换为十进制。

语法

```
hexdec(hex_string)
```

参数	描述
hex_string	必需。规定要转换的十六进制数。

说明

返回与 *hex_string* 参数所表示的十六进制数等值的的十进制数。hexdec() 将一个十六进制字符串转换为十进制数。所能转换的最大数值为 7fffffff，即十进制的 2147483647。PHP 4.1.0 开始，该函数可以处理大数字，这种情况下，它会返回 float 类型。

hexdec() 将遇到的所有非十六进制字符替换成 0。这样，所有左边的零都被忽略，但右边的零会计入值中。

例子

```
<?php
echo hexdec("1e");
echo hexdec("a");
echo hexdec("11ff");
echo hexdec("cceeff");
?>
```

输出：

```
30
10
4607
13430527
```

PHP hypot() 函数

定义和用法

hypot() 函数计算一直角三角形的斜边长度。

语法

```
hypot(x,y)
```

参数	描述
x	必需。边 x 的长度。
y	必需。边 y 的长度。

说明

hypot() 函数将会跟据直角三角形的两直解边长度 x 和 y 计算其斜边的长度。或者是从标点 (x , y) 到原点的距离。该函数的算法等同于 sqrt(x*x* + y*y*)。

例子

```
<?php
echo hypot(2,3);
echo hypot(3,6);
echo hypot(3,6);
echo hypot(1,3);
?>
```

输出：

```
3.60555127546
6.7082039325
6.7082039325
3.16227766017
```

PHP is_finite() 函数

定义和用法

is_finite() 函数判断是否为有限值。

语法

```
is_finite(x)
```

参数	描述
x	必需。规定要检查的值。

说明

如果 *x* 是本机平台上 PHP 浮点数所允许范围中的一个合法的有限值，则返回 true。

例子

```
<?php
echo is_finite(2);
echo is_finite(log(0));
echo is_finite(2000);
?>
```

输出：

```
1

1
```

PHP is_infinite() 函数

定义和用法

is_infinite() 判断是否为无限值。

语法

```
is_infinite(x)
```

参数	描述
x	必需。规定要检查的值。

说明

如果 *x* 为无穷大（正的或负的），例如 log(0) 的结果或者任何超出本平台的浮点数范围的值，则返回 true。

例子

```
<?php
echo is_infinite(2);
echo is_infinite(log(0));
echo is_infinite(2000);
?>
```

输出：

```
1
```

PHP is_nan() 函数

定义和用法

is_nan() 判断是否为合法数值。

语法

```
is_nan(x)
```

参数	描述
x	必需。规定要检查的值。

说明

如果 *x* 为“非数值”，例如 acos(1.01) 的结果，则返回 true。

例子

```
<?php
echo is_nan(200);
echo is_nan(acos(1.01));
?>
```

输出：

```
1
```

PHP lcg_value() 函数

定义和用法

lcg_value() 组合线性同余发生器。

语法

```
lcg_value()
```

说明

lcg_value() 返回范围为 (0, 1) 的一个伪随机数。本函数组合了周期为 2^31 - 85 和 2^31 - 249 的两个同余发生器。本函数的周期等于这两个素数的乘积。

例子

```
<?php
echo lcg_value();
?>
```

输出类似：

```
0.508212039328
```

PHP log() 函数

定义和用法

log() 返回自然对数。

语法

```
log(x,base)
```

参数	描述
x	必需。一个数。
base	可选。如果规定了该参数，则返回 log_{base}x。

说明

如果指定了可选的参数 *base*，log() 返回 log_{base}x ，否则 log() 返回参数 *x* 的自然对数。

注释：参数 base 自 PHP 4.3.0 开始可用。你可以计算任意以 b 为底 n 的对数，但其实使用的是数学等式：logb(n) = log(n)/log(b)，其中 log 是自然对数。

例子

```
<?php
echo lcg_value();
?>
```

输出类似：

```
0.508212039328
```

PHP log10() 函数

定义和用法

log10() 以 10 为底的对数。

语法

```
log10(x)
```

参数	描述
x	必需。一个数。

说明

返回参数 *x* 以 10 为底的对数。

提示和注释：

注释：如果参数 x 是负数，则返回 -1.#IND。

例子

在本例中，我们对不同的数应用 log10() 函数：

```
<?php
echo(log10(2.7183));
echo(log10(2));
echo(log10(1));
echo(log10(0));
echo(log10(-1));
?>
```

输出类似：

```
0.434297385125
0.301029995664
0
-1.#INF
-1.#IND
```

PHP log1p() 函数

定义和用法

log1p() 以返回 log(1 + x)，甚至当 x 的值接近零也能计算出准确结果。

语法

```
log1p(x)
```

参数	描述
x	必需。一个数。

说明

警告：本函数是实验性的。本函数的行为，包括函数名称以及其它任何关于本函数的文档可能会在没有通知的情况下随 PHP 以后的发布而改变。使用本函数风险自担。

PHP max() 函数

定义和用法

max() 返回最大值。

语法

```
max(x,y)
```

参数	描述
x	必需。一个数。
y	必需。一个数。

说明

max() 返回参数中数值最大的值。

如果仅有一个参数且为数组，max() 返回该数组中最大的值。如果第一个参数是整数、字符串或浮点数，则至少需要两个参数而 max() 会返回这些值中最大的一个。可以比较无限多个值。

提示和注释

注释：PHP 会将非数值的字符串当成 0，但如果这个正是最大的数值则仍然会返回一个字符串。如果多个参数都求值为 0 且是最大值，max() 会返回其中数值的 0，如果参数中没有数值的 0，则返回按字母表顺序最大的字符串。

例子

在本例中，我们将使用 max() 来返回两个指定的数中的最大值：

```
<?php
echo(max(5,7));
echo(max(-3,5));
echo(max(-3,-5));
echo(max(7.25,7.30));
?>
```

输出类似：

```
7
5
-3
7.3
```

PHP min() 函数

定义和用法

min() 返回最小值。

语法

```
min(x,y)
```

参数	描述
x	必需。一个数。
y	必需。一个数。

说明

min() 返回参数中数值最小的。

如果仅有一个参数且为数组，min() 返回该数组中最小的值。如果第一个参数是整数、字符串或浮点数，则至少需要两个参数而 min() 会返回这些值中最小的一个。可以比较无限多个值。

提示和注释

注释：PHP 会将非数值的 string 当成 0，但如果这个正是最小的数值则仍然会返回一个字符串。如果多个参数都求值为 0 且是最小值，min() 会返回按字母表顺序最小的字符串，如果其中没有字符串的话，则返回数值的 0。

例子

在本例中，我们将使用 min() 来返回两个指定的数中的最小值：

```
<?php
echo(min(5,7));
echo(min(-3,5));
echo(min(-3,-5));
echo(min(7.25,7.30));
?>
```

输出类似：

```
5
-3
-5
7.25
```

PHP mt_getrandmax() 函数

定义和用法

mt_getrandmax() 显示随机数的最大可能值。

语法

```
mt_getrandmax()
```

说明

返回调用 mt_rand() 所能返回的最大的随机数。

例子

```
<?php
echo mt_getrandmax();
?>
```

输出类似：

```
3147483649
```

PHP mt_rand() 函数

定义和用法

mt_rand() 使用 Mersenne Twister 算法返回随机整数。

语法

```
mt_rand(min,max)
```

说明

如果没有提供可选参数 *min* 和 *max*，mt_rand() 返回 0 到 RAND_MAX 之间的伪随机数。例如想要 5 到 15（包括 5 和 15）之间的随机数，用 mt_rand(5, 15)。

很多老的 libc 的随机数发生器具有一些不确定和未知的特性而且很慢。PHP 的 rand() 函数默认使用 libc 随机数发生器。mt_rand() 函数是非正式用来替换它的。该函数用了 Mersenne Twister 中已知的特性作为随机数发生器，它可以产生随机数值的平均速度比 libc 提供的 rand() 快四倍。

提示和注释

注释：自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 函数给随机数发生器播种，现在已自动完成。

注释：在 3.0.7 之前的版本中，max 的含义是 range 。要在这些版本中得到和上例相同 5 到 15 的随机数，简短的例子是 mt_rand (5, 11)。

例子

在本例中，我们会返回一些随机数：

```
<?php
echo(mt_rand());
echo(mt_rand());
echo(mt_rand(10,100));
?>
```

输出类似：

```
3150906288
513289678
35
```

PHP mt_srand() 函数

定义和用法

mt_srand() 播种 Mersenne Twister 随机数生成器。

语法

```
mt_srand(seed)
```

参数	描述
seed	必需。用 seed 来给随机数发生器播种。

说明

从 PHP 4.2.0 版开始，*seed* 参数变为可选项，当该项为空时，会被设为随时数。

提示和注释

注释：自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 函数给随机数发生器播种，现已自动完成。

例子

在本例中，我们将播种随机数生成器：

```
<?php
mt_srand(mktime());
echo(mt_rand());
?>
```

输出类似：

```
1132656473
```

PHP octdec() 函数

定义和用法

octdec() 函数把八进制转换为十进制。

语法

```
octdec(octal_string)
```

参数	描述
octal_string	必需。规定要转换的八进制数。

说明

返回 *octal_string* 参数所表示的八进制数的十进制等值。可转换的最大的数值为 17777777777 或十进制的 2147483647。从 PHP 4.1.0 开始，该函数可以处理大数字，这种情况下，它会返回 float 类型。

例子

```
<?php
echo octdec("36");
echo octdec("12");
echo octdec("3063");
echo octdec("106");
?>
```

输出类似：

```
30
10
1587
70
```

PHP pi() 函数

定义和用法

pi() 函数返回圆周率的值。

语法

```
pi()
```

说明

返回圆周率的近似值。返回值的 float 精度是由 php.ini 中的 precision 指令确定。默认值是 14。您也可以使用 M_PI 常量，该常量产生与 pi() 完全相同的结果。

例子

```
<?php
echo pi();
?>
```

输出类似：

```
3.14159265359
```

PHP pow() 函数

定义和用法

pow() 函数返回 x 的 y 次方。

语法

```
pow(x,y)
```

参数	描述
x	必需。一个数。
y	必需。一个数。

说明

返回 *x* 的 *y* 次方的幂。如果可能，本函数会返回 integer。

如果不能计算幂，将发出一条警告，pow() 将返回 false。PHP 4.2.0 版开始 pow() 不要产生任何的警告。

例子

```
<?php
echo pow(4,2);
echo pow(6,2);
echo pow(-6,2);
echo pow(-6,-2);
echo pow(-6,5.5);
?>
```

输出类似：

```
16
36
36
0.0277777777778
-1.#IND
```

PHP rad2deg() 函数

定义和用法

rad2deg() 函数把弧度数转换为角度数。

语法

```
rad2deg(radian_number)
```

参数	描述
radian_number	必需。规定要转换的弧度。

说明

本函数将 *radian_number* 从弧度转换为角度。

例子

```
<?php
$rad = M_PI;
$deg = rad2deg($rad);
echo "$rad radians is equal to $deg degrees";
?>
```

输出类似：

```
3.14159265359 radians is equal to 180 degrees
```

PHP rand() 函数

定义和用法

rand() 函数返回随机整数。

语法

```
rand(min,max)
```

参数	描述
min,max	可选。规定随机数产生的范围。

说明

如果没有提供可选参数 *min* 和 *max*，rand() 返回 0 到 RAND_MAX 之间的伪随机整数。例如，想要 5 到 15（包括 5 和 15）之间的随机数，用 rand(5, 15)。

提示和注释

注释：在某些平台下（例如 Windows）RAND_MAX 只有 32768。如果需要的范围大于 32768，那么指定 min 和 max 参数就可以生成大于 RAND_MAX 的数了，或者考虑用 mt_rand() 来替代它。

注释：自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 函数给随机数发生器播种，现在已自动完成。

注释：在 3.0.7 之前的版本中，max 的含义是 range 。要在这些版本中得到和上例相同 5 到 15 的随机数，简短的例子是 rand (5, 11)。

例子

本例会返回一些随机数：

```
<?php
echo(rand();
echo(rand();
echo(rand(10,100))
?>
```

输出：

```
17757
3794
97
```

PHP round() 函数

定义和用法

round() 函数对浮点数进行四舍五入。

语法

```
round(x,prec)
```

参数	描述
x	可选。规定要舍入的数字。
prec	可选。规定小数点后的位数。

说明

返回将 *x* 根据指定精度 *prec* （十进制小数点后数字的数目）进行四舍五入的结果。*prec* 也可以是负数或零（默认值）。

提示和注释

注释：PHP 默认不能正确处理类似 "12,300.2" 的字符串。

注释：*prec* 参数是在 PHP 4 中被引入的。。

例子

```
<?php
echo(round(0.60));
echo(round(0.50));
echo(round(0.49));
echo(round(-4.40));
echo(round(-4.60));
?>
```

输出：

```
1
1
0
-4
-5
```

PHP sin() 函数

定义和用法

sin() 函数返回一个数的正弦。

语法

```
sin(x)
```

参数	描述
x	必需。一个数。

说明

sin() 返回参数 *x* 的正弦值。参数 *x* 的单位为弧度。

提示和注释

注释：sin() 函数返回的数值介于 -1 和 1 之间。

例子

在本例中，我们将计算不同值的正弦：

```
<?php
echo(sin(3));
echo(sin(-3));
echo(sin(0));
echo(sin(M_PI));
echo(sin(M_PI_2))
?>
```

输出：

```
0.14112000806
-0.14112000806
0
1.22460635382E-016
1
```

PHP sinh() 函数

定义和用法

sinh() 函数返回一个数的双曲正弦。

语法

```
sinh(x)
```

参数	描述
x	必需。一个数字。

说明

返回 *x* 的双曲正弦值，定义为 (exp(arg) - exp(-arg))/2。

例子

在本例中，我们将返回不同数的双曲正弦：

```
<?php
echo(sinh(3));
echo(sinh(-3));
echo(sinh(0));
echo(sinh(M_PI));
echo(sinh(M_PI_2));
?>
```

输出：

```
10.0178749274
-10.0178749274
0
11.5487393573
2.30129890231
```

PHP sqrt() 函数

定义和用法

sqrt() 函数返回一个数的平方根。

语法

```
sqrt(x)
```

参数	描述
x	必需。一个数字。

说明

返回 *x* 的平方根。

提示和注释

注释：如果参数 *x* 是负数，则 sqrt() 函数返回 -1.#IND。

例子

在本例中，我们将返回不同数的平方根：

```
<?php
echo(sqrt(0));
echo(sqrt(1));
echo(sqrt(9));
echo(sqrt(0.64));
echo(sqrt(-9));
?>
```

输出：

```
0
1
3
0.8
-1.#IND
```

PHP srand() 函数

定义和用法

srand() 函数播下随机数发生器种子。

语法

```
srand(seed)
```

参数	描述
seed	可选。用 seed 播下随机数发生器种子。

说明

从 PHP 4.2.0 版开始，*seed* 参数变为可选项，当该项为空时，会被设为随时数。

提示和注释

注释：自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 函数给随机数发生器播种，现在已自动完成。

例子

在本例中，我们将播种随机数发生器：

```
<?php
srand(mktime());
echo(rand());
?>
```

输出：

```
23054
```

PHP tan() 函数

定义和用法

tan() 函数返回正切。

语法

```
tan(x)
```

参数	描述
x	必需。一个数。

说明

tan() 返回参数 *x* 的正切值。参数 *x* 的单位为弧度

例子

在本例中，我们将返回不同的数的正切：

```
<?php
echo(tan(M_PI_4));
echo(tan(0.50));
echo(tan(-0.50));
echo(tan(5));
echo(tan(10));
echo(tan(-5));
echo(tan(-10));
?>
```

输出：

```
1
0.546302489844
-0.546302489844
-3.38051500625
0.648360827459
3.38051500625
-0.648360827459
```

PHP tanh() 函数

定义和用法

tanh() 函数返回双曲正切。

语法

```
tanh(x)
```

参数	描述
x	必需。一个数。

说明

返回 *x* 的双曲正切值，定义为 sinh(arg)/cosh(arg)。

例子

在本例中，我们将返回不同的数的双曲正切：

```
<?php
echo(tanh(M_PI_4));
echo(tanh(0.50));
echo(tanh(-0.50));
echo(tanh(5));
echo(tanh(10));
echo(tanh(-5));
echo(tanh(-10))
?>
```

输出：

```
0.655794202633
0.46211715726
-0.46211715726
0.999909204263
0.999999995878
-0.999909204263
-0.999999995878
```

PHP 5 MySQLi 函数

PHP MySQLi 简介

PHP MySQLi = PHP MySQL Improved!

MySQLi 函数允许您访问 MySQL 数据库服务器。

注释：MySQLi 扩展被设计用于 MySQL 4.1.13 版本或更新的版本。

安装 / Runtime 配置

为了能够顺利使用 MySQLi 函数，您必须在编译 PHP 时添加对 MySQLi 扩展的支持。

MySQLi 扩展是在 PHP 5.0.0 版本中引进的。MySQL Native Driver 包含在 PHP 5.3.0 版本。

有关安装的详细信息，请访问：http://www.php.net/manual/en/mysqli.installation.php

有关运行配置的详细信息，请访问：http://www.php.net/manual/en/mysqli.configuration.php

PHP 5 MySQLi 函数

函数	描述
mysqli_affected_rows()	返回前一次 MySQL 操作所影响的记录行数。
mysqli_autocommit()	打开或关闭自动提交数据库修改。
mysqli_change_user()	更改指定数据库连接的用户。
mysqli_character_set_name()	返回数据库连接的默认字符集。
mysqli_close()	关闭先前打开的数据库连接。
mysqli_commit()	提交当前事务。
mysqli_connect_errno()	返回上一次连接错误的错误代码。
mysqli_connect_error()	返回上一次连接错误的错误描述。
mysqli_connect()	打开一个到 MySQL 服务器的新的连接。
mysqli_data_seek()	调整结果指针到结果集中的一个任意行。
mysqli_debug()	执行调试操作。
mysqli_dump_debug_info()	转储调试信息到日志中。
mysqli_errno()	返回最近调用函数的最后一个错误代码。

mysqli_error_list()	返回最近调用函数的错误列表。
mysqli_error()	返回最近调用函数的最后一个错误描述。
mysqli_fetch_all()	从结果集中取得所有行作为关联数组，或数字数组，或二者兼有。
mysqli_fetch_array()	从结果集中取得一行作为关联数组，或数字数组，或二者兼有。
mysqli_fetch_assoc()	从结果集中取得一行作为关联数组。
mysqli_fetch_field_direct()	从结果集中取得某个单一字段的 meta-data，并作为对象返回。
mysqli_fetch_field()	从结果集中取得下一字段，并作为对象返回。
mysqli_fetch_fields()	返回结果中代表字段的对象的数组。
mysqli_fetch_lengths()	返回结果集中当前行的每个列的长度。
mysqli_fetch_object()	从结果集中取得当前行，并作为对象返回。
mysqli_fetch_row()	从结果集中取得一行，并作为枚举数组返回。
mysqli_field_count()	返回最近查询的列数。
mysqli_field_seek()	把结果集中的指针设置为指定字段的偏移量。
mysqli_field_tell()	返回结果集中的指针的位置。
mysqli_free_result()	释放结果内存。
mysqli_get_charset()	返回字符集对象。
mysqli_get_client_info()	返回 MySQL 客户端库版本。
mysqli_get_client_stats()	返回有关客户端每个进程的统计。
mysqli_get_client_version()	将 MySQL 客户端库版本作为整数返回。
mysqli_get_connection_stats()	返回有关客户端连接的统计。
mysqli_get_host_info()	返回 MySQL 服务器主机名和连接类型。
mysqli_get_proto_info()	返回 MySQL 协议版本。
mysqli_get_server_info()	返回 MySQL 服务器版本。
mysqli_get_server_version()	将 MySQL 服务器版本作为整数返回。
mysqli_info()	返回有关最近执行查询的信息。
mysqli_init()	初始化 MySQLi 并返回 mysqli_real_connect() 使用的资源。
mysqli_insert_id()	返回最后一个查询中自动生成的 ID。
mysql_kill()	请求服务器杀死一个 MySQL 线程。
mysqli_more_results()	检查一个多查询是否有更多的结果。

mysqli_multi_query()	执行一个或多个针对数据库的查询。
mysqli_next_result()	为 mysqli_multi_query() 准备下一个结果集。
mysqli_num_fields()	返回结果集中字段的数量。
mysqli_num_rows()	返回结果集中行的数量。
mysqli_options()	设置额外的连接选项，用于影响连接行为。
mysqli_ping()	进行一个服务器连接，如果连接已断开则尝试重新连接。
mysqli_prepare()	准备执行一个 SQL 语句。
mysqli_query()	执行某个针对数据库的查询。
mysqli_real_connect()	打开一个到 MySQL 服务器的新的链接。
mysqli_real_escape_string()	转义在 SQL 语句中使用的字符串中的特殊字符。
mysqli_real_query()	执行 SQL 查询
mysqli_reap_async_query()	返回异步查询的结果。
mysqli_refresh()	刷新表或缓存，或者重置复制服务器信息。
mysqli_rollback()	回滚数据库中的当前事务。
mysqli_select_db()	更改连接的默认数据库。
mysqli_set_charset()	设置默认客户端字符集。
mysqli_set_local_infile_default()	撤销用于 load local infile 命令的用户自定义句柄。
mysqli_set_local_infile_handler()	设置用于 LOAD DATA LOCAL INFILE 命令的回滚函数。
mysqli_sqlstate()	返回最后一个 MySQL 操作的 SQLSTATE 错误代码。
mysqli_ssl_set()	用于创建 SSL 安全连接。
mysqli_stat()	返回当前系统状态。
mysqli_stmt_init()	初始化声明并返回 mysqli_stmt_prepare() 使用的对象。
mysqli_store_result()	传输最后一个查询的结果集。
mysqli_thread_id()	返回当前连接的线程 ID。
mysqli_thread_safe()	返回是否将客户端库编译成 thread-safe。
mysqli_use_result()	从上次使用 mysqli_real_query() 执行的查询中初始化结果集的检索。
mysqli_warning_count()	返回连接中的最后一个查询的警告数量。

PHP mysqli_affected_rows() 函数

实例

从不同的查询中输出所影响记录行数：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Perform queries and print out affected rows
mysqli_query($con,"SELECT * FROM Persons");
echo "Affected rows: " . mysqli_affected_rows($con);

mysqli_query($con,"DELETE FROM Persons WHERE Age>32");
echo "Affected rows: " . mysqli_affected_rows($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_affected_rows() 函数返回前一次 MySQL 操作（SELECT、INSERT、UPDATE、REPLACE、DELETE）所影响的记录行数。

语法

mysqli*affected_rows(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	一个 **> 0** 的整数表示所影响的记录行数。**0** 表示没有受影响的记录。**-1** 表示查询返回错误。
PHP 版本：	5+
:--	---

PHP mysqli_autocommit() 函数

实例

关闭自动提交，做一些查询，然后提交查询：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Set autocommit to off
mysqli_autocommit($con,FALSE);

// Insert some values
mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Peter','Griffin',35)");
mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Glenn','Quagmire',33)");

// Commit transaction
mysqli_commit($con);

// Close connection
mysqli_close($con);
?>
```

定义和用法

mysqli_autocommit() 函数开启或关闭自动提交数据库修改。

提示：请查看 mysqli_commit() 函数，用于提交指定数据库连接的当前事务。请查看 mysqli_rollback() 函数，用于回滚当前事务。

语法

mysqli*autocommit(_connection,mode);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。
mode	必需。如果设置为 FALSE，则表示关闭 auto-commit。如果设置为 TRUE，则表示开启 auto-commit（提交任何等待查询）。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_change_user() 函数

实例

改变指定数据库连接的用户：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");

// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Reset all and select a new database
mysqli_change_user($link, "my_user", "my_password", "my_test");

mysqli_close($con);
?>
```

定义和用法

mysqli_change_user() 函数改变指定数据库连接的用户，并设置当前数据库。

语法

mysqli*change_user(_connection,username,password,dbname);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。
username	必需。规定 MySQL 用户名。
password	必需。规定 MySQL 密码。
dbname	必需。规定要改变的新数据库。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_character_set_name() 函数

实例

返回数据库连接的默认字符集：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$charset=mysqli_character_set_name($con);
echo "Default character set is: " . $charset;

mysqli_close($con);
?>
```

定义和用法

mysqli_character_set_name() 函数返回数据库连接的默认字符集。

语法

mysqli*character_set_name(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	指定连接的默认字符集。
PHP 版本：	5+
:--	---

PHP mysqli_close() 函数

实例

关闭先前打开的数据库连接：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");

// ....some PHP code...

mysqli_close($con);
?>
```

定义和用法

mysqli_close() 函数关闭先前打开的数据库连接。

语法

mysqli*close(_connection*);

参数	描述
connection	必需。规定要关闭的 MySQL 连接。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_commit() 函数

实例

关闭自动提交，做一些查询，然后提交查询：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Set autocommit to off
mysqli_autocommit($con,FALSE);

// Insert some values
mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Peter','Griffin',35)");
mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Glenn','Quagmire',33)");

// Commit transaction
mysqli_commit($con);

// Close connection
mysqli_close($con);
?>
```

定义和用法

mysqli_commit() 函数提交指定数据库连接的当前事务。

提示：请查看 mysqli_autocommit() 函数，用于开启或关闭自动提交数据库修改。请查看 mysqli_rollback() 函数，用于回滚当前事务。

语法

mysqli*commit(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_connect_errno() 函数

实例

返回上一次连接错误的错误代码：

```
<?php
$con=mysqli_connect("localhost","wrong_user","my_password","my_db");
// Check connection
if (!$con)
{
die("Connection error: " . mysqli_connect_errno();
}
?>
```

定义和用法

mysqli_connect_errno() 函数返回上一次连接错误的错误代码。

语法

mysqli*connect_errno()*;_

技术细节

返回值：	返回错误代码值，如果没有错误发生则返回 **0**。
PHP 版本：	5+
:--	---

PHP mysqli_connect_error() 函数

实例

返回上一次连接错误的错误描述：

```
<?php
$con=mysqli_connect("localhost","wrong_user","my_password","my_db");
// Check connection
if (!$con)
{
die("Connection error: " . mysqli_connect_error();
}
?>
```

定义和用法

mysqli_connect_error() 函数返回上一次连接错误的错误描述。

语法

mysqli*connect_error()*;_

技术细节

返回值：	返回一个描述错误的字符串。如果没有错误发生则返回 **NULL**。
PHP 版本：	5+
:--	---

PHP mysqli_connect() 函数

实例

打开一个到 MySQL 服务器的新的连接：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");

// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
?>
```

定义和用法

mysqli_connect() 函数打开一个到 MySQL 服务器的新的连接。

语法

mysqli*connect(_host,username,password,dbname,port,socket*);

参数	描述
host	可选。规定主机名或 IP 地址。
username	可选。规定 MySQL 用户名。
password	可选。规定 MySQL 密码。
dbname	可选。规定默认使用的数据库。
port	可选。规定尝试连接到 MySQL 服务器的端口号。
socket	可选。规定 socket 或要使用的已命名 pipe。

技术细节

返回值：	返回一个代表到 **MySQL** 服务器的连接的对象。
PHP 版本：	5+
:--	---

PHP mysqli_data_seek() 函数

实例

在结果集中寻找行号 15：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Seek to row number 15
mysqli_data_seek($result,14);

// Fetch row
$row=mysqli_fetch_row($result);

printf ("Lastname: %s Age: %sn", $row[0], $row[1]);

// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_data_seek() 函数调整结果指针到结果集中的一个任意行。

语法

mysqli*data_seek(_result,offset*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。
offset	必需。规定字段偏移。范围必须在 0 和 行总数 - 1 之间。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_debug() 函数

实例

在本地机上的 "/temp/client.trace" 中创建一个 trace 文件：

```
<?php
mysqli_debug("d:t:o,/temp/client.trace");
?>
```

定义和用法

mysqli_debug() 函数用于执行调试操作。

注释：为了使用该函数，您必须编译 MySQL 客户端库来支持调试。

语法

mysqli*debug(_message);*

参数	描述
message	必需。一个代表要执行的调试操作的字符串。

技术细节

返回值：	**TRUE**
PHP 版本：	5+
:--	---

PHP mysqli_dump_debug_info() 函数

实例

转储调试信息到日志中：

```
<?php
mysqli_dump_debug_info($con);
?>
```

定义和用法

mysqli_dump_debug_info() 函数转储调试信息到日志中。

语法

mysqli*dump_debug_info(_link);*

参数	描述
link	必需。一个由 mysqli_connect() 或 mysqli_init() 返回的连接标识符。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_errno() 函数

实例

返回最近调用函数的最后一个错误代码：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Perform a query, check for error
if (!mysqli_query($con,"INSERT INTO Persons (FirstName) VALUES ('Glenn')"))
{
echo("Errorcode: " . mysqli_errno($con));
}

mysqli_close($con);
?>
```

定义和用法

mysqli_errno() 函数返回最近调用函数的最后一个错误代码。

语法

mysqli*errno(_connection);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回错误代码值。如果没有错误发生则返回 **0**。
PHP 版本：	5+
:--	---

PHP mysqli_error_list() 函数

实例

返回最近调用函数的错误列表：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Perform a query, check for error
if (!mysqli_query($con,"INSERT INTO Persons (FirstName) VALUES ('Glenn')"))
{
print_r(mysqli_error_list($con));
}

mysqli_close($con);
?>
```

定义和用法

mysqli_error_list() 函数返回最近调用函数的错误列表。

语法

mysqli*error_list(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回错误列表。每个错误都是一个带有 **errno**（错误代码）、**error**（错误文本）和 **sqlstate** 的关联数组。
PHP 版本：	5.4+
:--	---

PHP mysqli_error() 函数

实例

返回最近调用函数的最后一个错误描述：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Perform a query, check for error
if (!mysqli_query($con,"INSERT INTO Persons (FirstName) VALUES ('Glenn')"))
{
echo("Error description: " . mysqli_error($con));
}

mysqli_close($con);
?>
```

定义和用法

mysqli_error() 函数返回最近调用函数的最后一个错误描述。

语法

mysqli*error(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个带有错误描述的字符串。如果没有错误发生则返回 ""。
PHP 版本：	5+
:--	---

PHP mysqli_fetch_all() 函数

实例

从结果集中取得所有行作为关联数组：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";
$result=mysqli_query($con,$sql);

// Fetch all
mysqli_fetch_all($result,MYSQLI_ASSOC);

// Free result set
mysqli_free_result($result);

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_all() 函数从结果集中取得所有行作为关联数组，或数字数组，或二者兼有。

注释：该函数只在带有 MySQL Native Driver 时可用。

语法

mysqli*fetch_all(_result,resulttype);*

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。
resulttype	可选。规定应该产生哪种类型的数组。可以是以下值中的一个：`MYSQLI_ASSOC` `MYSQLI_NUM` `MYSQLI_BOTH`

技术细节

返回值：	返回包含结果行的关联数组或数字数组。
PHP 版本：	5.3+
:--	---

PHP mysqli_fetch_array() 函数

实例

从结果集中取得一行作为数字数组或关联数组：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";
$result=mysqli_query($con,$sql)

// Numeric array
$row=mysqli_fetch_array($result,MYSQLI_NUM);
printf ("%s (%s)n",$row[0],$row[1]);

// Associative array
$row=mysqli_fetch_array($result,MYSQLI_ASSOC);
printf ("%s (%s)n",$row["Lastname"],$row["Age"]);

// Free result set
mysqli_free_result($result);

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_array() 函数从结果集中取得一行作为关联数组，或数字数组，或二者兼有。

注释：该函数返回的字段名是区分大小写的。

语法

mysqli*fetch_array(_result,resulttype*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。
resulttype	可选。规定应该产生哪种类型的数组。可以是以下值中的一个：`MYSQLI_ASSOC` `MYSQLI_NUM` `MYSQLI_BOTH`

技术细节

返回值：	返回与读取行匹配的字符串数组。如果结果集中没有更多的行则返回 **NULL**。
PHP 版本：	5+
:--	---

PHP mysqli_fetch_assoc() 函数

实例

从结果集中取得一行作为关联数组：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";
$result=mysqli_query($con,$sql)

// Associative array
$row=mysqli_fetch_assoc($result);
printf ("%s (%s)n",$row["Lastname"],$row["Age"]);

// Free result set
mysqli_free_result($result);

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_array() 函数从结果集中取得一行作为关联数组。

注释：该函数返回的字段名是区分大小写的。

语法

mysqli*fetch_assoc(_result*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	返回代表读取行的关联数组。如果结果集中没有更多的行则返回 **NULL**。
PHP 版本：	5+
:--	---

PHP mysqli_fetch_field_direct() 函数

实例

返回结果集中某个单一字段（列）的 meta-data，并输出字段名称、表格和最大长度：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Get field information for "Age"
$fieldinfo=mysqli_fetch_field_direct($result,1);

printf("Name: %sn",$fieldinfo->name);
printf("Table: %sn",$fieldinfo->table);
printf("max. Len: %dn",$fieldinfo->max_length);

// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_field_direct() 函数从结果集中取得某个单一字段（列）的 meta-data，并作为对象返回。

语法

mysqli*fetch_field_direct(_result,fieldnr);*

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。
fieldnr	必需。规定字段号。必须介于 0 和 字段数-1 之间。

技术细节

返回值：	返回包含字段定义信息的对象。如果没有可用信息则返回 **FALSE**。该对象有下列属性： `name` - 列名 `orgname` - 原始的列名（如果指定了别名） `table` - 表名 `orgtable` - 原始的表名（如果指定了别名） `def` - 该字段的默认值 `max_length` - 字段的最大宽度 `length` - 在表定义中规定的字段宽度 `charsetnr` - 字段的字符集号 `flags` - 字段的位标志 `type` - 用于字段的数据类型 `decimals` - 整数字段，小数点后的位数
PHP 版本：	5+
:--	---

PHP mysqli_fetch_field() 函数

实例

返回结果集中下一字段（列），然后输出每个字段名称、表格和最大长度：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Get field information for all fields
while ($fieldinfo=mysqli_fetch_field($result))
{
printf("Name: %sn",$fieldinfo->name);
printf("Table: %sn",$fieldinfo->table);
printf("max. Len: %dn",$fieldinfo->max_length);
}
// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_field() 函数从结果集中取得下一字段（列），并作为对象返回。

语法

mysqli*fetch_field(_result*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	返回包含字段定义信息的对象。如果没有可用信息则返回 **FALSE**。该对象有下列属性： `name` - 列名 `orgname` - 原始的列名（如果指定了别名） `table` - 表名 `orgtable` - 原始的表名（如果指定了别名） `def` - 保留作为默认值，当前总是为 **""** `db` - 数据库（在 **PHP 5.3.6** 中新增的） `catalog` - 目录名称，总是为 **"def"**（自 **PHP 5.3.6** 起） `max_length` - 字段的最大宽度 `length` - 在表定义中规定的字段宽度 `charsetnr` - 字段的字符集号 `flags` - 字段的位标志 `type` - 用于字段的数据类型 `decimals` - 整数字段，小数点后的位数
PHP 版本：	5+
:--	---

PHP mysqli_fetch_fields() 函数

实例

返回结果集中代表字段（列）的对象的数组，然后输出每个字段名称、表格和最大长度：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Get field information for all fields
$fieldinfo=mysqli_fetch_fields($result);

foreach ($fieldinfo as $val)
{
printf("Name: %sn",$val->name);
printf("Table: %sn",$val->table);
printf("max. Len: %dn",$val->max_length);
}
// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_fields() 函数返回结果集中代表字段（列）的对象的数组。

语法

mysqli*fetch_fields(_result);*

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	返回包含字段定义信息的对象。如果没有可用信息则返回 **FALSE**。该对象有下列属性： `name` - 列名 `orgname` - 原始的列名（如果指定了别名） `table` - 表名 `orgtable` - 原始的表名（如果指定了别名） `max_length` - 字段的最大宽度 `length` - 在表定义中规定的字段宽度 `charsetnr` - 字段的字符集号 `flags` - 字段的位标志 `type` - 用于字段的数据类型 `decimals` - 整数字段，小数点后的位数
PHP 版本：	5+
:--	---

PHP mysqli_fetch_lengths() 函数

实例

返回结果集中的字段长度：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT * FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
$row=mysqli_fetch_row($result);

// Display field lengths
foreach (mysqli_fetch_lengths($result) as $i=>$val)
{
printf("Field %2d has length: %2dn",$i+1,$val);
}

// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_lengths() 函数返回结果集中的字段长度。

语法

mysqli*fetch_lengths(_result*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	如果整数表示每个字段（列）的长度则返回数组，如果发生错误则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_fetch_object() 函数

实例

返回结果集中的当前行，然后输出每个字段的值：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
while ($obj=mysqli_fetch_object($result))
{
printf("%s (%s)n",$obj->Lastname,$obj->Age);
}
// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_object() 函数从结果集中取得当前行，并作为对象返回。

注释：该函数返回的字段名是区分大小写的。

语法

mysqli*fetch_object(_result,classname,params);*

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。
classname	可选。规定要实例化的类名称，设置属性并返回。
params	可选。规定一个传给 *classname* 对象构造器的参数数组。

技术细节

返回值：	返回带有所取得行的字符串属性的对象。如果在结果集中没有更多的行则返回 **NULL**。
PHP 版本：	5+
:--	---
更新日志：	在 PHP 5.0.0 中新增了作为不同对象返回的功能。
:--	---

PHP mysqli_fetch_row() 函数

实例

从结果集中取得行：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Fetch one and one row
while ($row=mysqli_fetch_row($result))
{
printf ("%s (%s)n",$row[0],$row[1]);
}
// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_fetch_row() 函数从结果集中取得一行，并作为枚举数组返回。

语法

mysqli*fetch_row(_result*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	返回一个与所取得行相对应的字符串数组。如果在结果集中没有更多的行则返回 **NULL**。
PHP 版本：	5+
:--	---

PHP mysqli_field_count() 函数

实例

假设我们有一个 "Friends" 表，其中有 3 个字段 20 行记录。返回最近查询的列数：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_query($con,"SELECT * FROM Friends");
// Get number of columns - will always return 3
mysqli_field_count($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_field_count() 函数返回最近查询的列数。

语法

mysqli*field_count(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个表示结果集中列数的整数。
PHP 版本：	5+
:--	---

PHP mysqli_field_seek() 函数

实例

设置结果集中第一个字段（列）的字段指针，然后通过 mysqli_fetch_field() 获取字段信息并输出字段名称、表格和最大长度：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Get field info for 1st column ("Lastname")
mysqli_field_seek($result,0);
$fieldinfo=mysqli_fetch_field($result);

printf("Name: %sn",$fieldinfo->name);
printf("Table: %sn",$fieldinfo->table);
printf("max. Len: %dn",$fieldinfo->max_length);

// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_field_seek() 函数把字段指针设置为指定字段的偏移量。

语法

mysqli*field_seek(_result,fieldnr*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。
fieldnr	必需。规定字段号。必须介于 0 和 字段数-1 之间。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_field_tell() 函数

实例

取得所有字段的字段信息，然后通过 mysqli_field_tell() 取得当前字段并输出字段名称、表格和最大长度：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{

// Get field info for all fields
while ($fieldinfo=mysqli_fetch_field($result))
{
// Get field cursor position
$currentfield=mysqli_field_tell($result);

printf("Column %d:n", $currentfield);
printf("Name: %sn", $fieldinfo->name);
printf("Table: %sn", $fieldinfo->table);
}

// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_field_tell() 函数返回字段指针的位置。

语法

mysqli*field_tell(_result*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	返回字段指针的当前偏移量。
PHP 版本：	5+
:--	---

PHP mysqli_free_result() 函数

实例

从结果集中取得行，然后释放结果内存：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Fetch one and one row
while ($row=mysqli_fetch_row($result))
{
printf ("%s (%s)n",$row[0],$row[1]);
}
// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_free_result() 函数释放结果内存。

语法

mysqli*free_result(_result*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	没有返回值。
PHP 版本：	5+
:--	---

PHP mysqli_get_charset() 函数

实例

返回带有属性的字符集对象：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

var_dump(mysqli_get_charset($con));

mysqli_close($con);
?>
```

定义和用法

mysqli_get_charset() 函数返回字符集对象。

语法

mysqli*get_charset(_connection);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回带有下列属性的字符集对象： `charset` - 字符集名称 `collation` - 排序规则名称 `dir` - 被获取的目录字符集或者 "" `min_length` - 以字节计的最小字符长度 `max_length` - 以字节计的最大字符长度 `number` - 内部字符集数 `state` - 字符集状态
PHP 版本：	5.1+
:--	---

PHP mysqli_get_client_info() 函数

实例

返回 MySQL 客户端库版本：

```
<?php
echo mysqli_get_client_info();
?>
```

定义和用法

mysqli_get_client_info() 函数返回 MySQL 客户端库版本。

语法

mysqli*get_client_info(_connection*);

参数	描述
connection	可选。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个表示 **MySQL** 客户端库版本的字符串。
PHP 版本：	5+
:--	---

PHP mysqli_get_client_stats() 函数

实例

返回有关客户端每个进程的统计：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

print_r(mysqli_get_client_stats());

mysqli_close($con);
?>
```

定义和用法

mysqli_get_client_stats() 函数返回有关客户端每个进程的统计。

语法

mysqli*get_client_stats()*;_

技术细节

返回值：	如果成功则返回一个带有客户端统计的数组，如果失败则返回 **FALSE**。
PHP 版本：	5.3+
:--	---

PHP mysqli_get_client_version() 函数

实例

将 MySQL 客户端库版本作为整数返回：

```
<?php
echo mysqli_get_client_version();
?>
```

定义和用法

mysqli_get_client_version() 函数将 MySQL 客户端库版本作为整数返回。

MySQL 客户端库版本将按照以下格式返回： 主要版本*10000* + 次要版本100 + 子版本。例如：5.1.0 将返回 50100。

语法

mysqli*get_client_version(_connection*);

参数	描述
connection	可选。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个表示 **MySQL** 客户端库版本的整数。
PHP 版本：	5+
:--	---

PHP mysqli_get_connection_stats() 函数

实例

返回有关客户端连接的统计：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

print_r(mysqli_get_connection_stats($con));

mysqli_close($con);
?>
```

定义和用法

mysqli_get_connection_stats() 函数返回有关客户端连接的统计。

语法

mysqli*get_connection_stats(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	如果成功则返回一个带有连接统计的数组，如果失败则返回 **FALSE**。
PHP 版本：	5.3+
:--	---

PHP mysqli_get_connection_stats() 函数

实例

返回有关客户端连接的统计：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

print_r(mysqli_get_connection_stats($con));

mysqli_close($con);
?>
```

定义和用法

mysqli_get_connection_stats() 函数返回有关客户端连接的统计。

语法

mysqli*get_connection_stats(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	如果成功则返回一个带有连接统计的数组，如果失败则返回 **FALSE**。
PHP 版本：	5.3+
:--	---

PHP mysqli_get_host_info() 函数

实例

返回 MySQL 服务器主机名和连接类型：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

echo mysqli_get_host_info($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_get_host_info() 函数返回 MySQL 服务器主机名和连接类型。

语法

mysqli*get_host_info(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个表示 **MySQL** 服务器主机名和连接类型的字符串。
PHP 版本：	5+
:--	---

PHP mysqli_get_proto_info() 函数

实例

返回 MySQL 协议版本：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

echo mysqli_get_proto_info($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_get_proto_info() 函数返回 MySQL 协议版本。

语法

mysqli*get_proto_info(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个表示 **MySQL** 协议版本的整数。
PHP 版本：	5+
:--	---

PHP mysqli_get_server_info() 函数

实例

返回 MySQL 服务器版本：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

echo mysqli_get_server_info($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_get_server_info() 函数返回 MySQL 服务器版本。

语法

mysqli*get_server_info(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个表示 **MySQL** 服务器版本的字符串。
PHP 版本：	5+
:--	---

PHP mysqli_get_server_version() 函数

实例

将 MySQL 服务器版本作为整数返回：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

echo mysqli_get_server_version($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_get_server_version() 函数将 MySQL 服务器版本作为整数返回。

MySQL 服务器版本将按照以下格式返回： 主要版本*10000* + 次要版本100 + 子版本。例如：5.1.0 将返回 50100。

语法

mysqli*get_server_version(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个表示 **MySQL** 服务器版本的整数。
PHP 版本：	5+
:--	---

PHP mysqli_info() 函数

实例

返回有关最近执行查询的信息：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Perform queries
$sql1="CREATE TABLE testPersons LIKE Persons"
mysqli_query($con,$sql1);
$sql2="INSERT INTO testPersons SELECT * FROM Persons ORDER BY LastName LIMIT 10"
mysqli_query($con,$sql2);

// Print info about most recently executed query
echo mysqli_info($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_info() 函数返回有关最近执行查询的信息。

该函数作用于下列查询类型：

- INSERT INTO...SELECT...
- INSERT INTO...VALUES (...),(...),(...)
- LOAD DATA INFILE ...
- ALTER TABLE ...
- UPDATE ...

语法

mysqli*info(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个字符串，包含有关最近执行查询的额外信息。
PHP 版本：	5+
:--	---

PHP mysqli_init() 函数

实例

mysqli_init() 函数的使用：

```
<?php
$con=mysqli_init();
if (!$con)
{
die("mysqli_init failed");
}

if (!mysqli_real_connect($con,"localhost","my_user","my_password","my_db"))
{
die("Connect Error: " . mysqli_connect_error());
}

mysqli_close($con);
?>
```

定义和用法

mysqli_init() 函数初始化 MySQLi 并返回 mysqli_real_connect() 使用的对象。

语法

mmysqli_init();

技术细节

返回值：	返回一个对象。
PHP 版本：	5+
:--	---

PHP mysqli_insert_id() 函数

实例

假设 Persons 表有一个自动生成的 ID 字段。返回最后一次查询中的 ID：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Glenn','Quagmire',33)");

// Print auto-generated id
echo "New record has id: " . mysqli_insert_id($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_insert_id() 函数返回最后一个查询中自动生成的 ID（通过 AUTO_INCREMENT 生成）。

语法

mysqli*insert_id(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个在最后一个查询中自动生成的带有 **AUTO_INCREMENT** 字段值的整数。如果数字 **>** 最大整数值，它将返回一个字符串。如果没有更新或没有 **AUTO_INCREMENT** 字段，将返回 **0**。
PHP 版本：	5+
:--	---

PHP mysqli_kill() 函数

实例

返回当前连接的线程 ID，然后杀死连接：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Get thread id
$t_id=mysqli_thread_id($con);

// Kill connection
mysqli_kill($con,$t_id);
?>
```

定义和用法

mysqli*kill()* 函数请求服务器杀死一个由 *_processid* 参数指定的 MySQL 线程。

语法

mysqli*kill(_connection,processid*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。
processid	必需。由 mysqli_thread_id() 返回的线程 ID。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_more_results() 函数

定义和用法

mysqli_more_results() 函数检查一个多查询是否有更多的结果。

语法

mysqli*more_results(_connection);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_multi_query() 函数

实例

执行多个针对数据库的查询：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql = "SELECT Lastname FROM Persons ORDER BY LastName;";
$sql .= "SELECT Country FROM Customers";

// Execute multi query
if (mysqli_multi_query($con,$sql))
{
do
{
// Store first result set
if ($result=mysqli_store_result($con))
{
while ($row=mysqli_fetch_row($result))
{
printf("%sn",$row[0]);
}
mysqli_free_result($con);
}
}
while (mysqli_next_result($con));
}

mysqli_close($con);
?>
```

定义和用法

mysqli_multi_query() 函数执行一个或多个针对数据库的查询。多个查询用分号进行分隔。

语法

mysqli*multi_query(_connection,query*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。
query	必需。规定一个或多个查询，用分号进行分隔。

技术细节

返回值：	如果第一个查询失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_next_result() 函数

实例

执行多个针对数据库的查询。请使用 mysqli_next_result() 函数来准备下一个结果集：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql = "SELECT Lastname FROM Persons ORDER BY LastName;";
$sql .= "SELECT Country FROM Customers";

// Execute multi query
if (mysqli_multi_query($con,$sql))
{
do
{
// Store first result set
if ($result=mysqli_store_result($con))
{
while ($row=mysqli_fetch_row($result))
{
printf("%sn",$row[0]);
}
mysqli_free_result($con);
}
}
while (mysqli_next_result($con));
}

mysqli_close($con);
?>
```

定义和用法

mysqli_next_result() 函数为 mysqli_multi_query() 准备下一个结果集。

语法

mysqli*next_result(_connection);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_num_fields() 函数

实例

返回结果集中字段（列）的数量：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Return the number of fields in result set
$fieldcount=mysqli_num_fields($result);
printf("Result set has %d fields.n",$fieldcount);
// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_num_fields() 函数返回结果集中字段（列）的数量。

语法

mysqli*num_fields(_result*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	返回结果集中字段的数量。
PHP 版本：	5+
:--	---

PHP mysqli_num_rows() 函数

实例

返回结果集中行的数量：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

if ($result=mysqli_query($con,$sql))
{
// Return the number of rows in result set
$rowcount=mysqli_num_rows($result);
printf("Result set has %d rows.n",$rowcount);
// Free result set
mysqli_free_result($result);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_num_rows() 函数返回结果集中行的数量。

语法

mysqli*num_rows(_result*);

参数	描述
result	必需。规定由 mysqli_query()、mysqli_store_result() 或 mysqli_use_result() 返回的结果集标识符。

技术细节

返回值：	返回结果集中行的数量。
PHP 版本：	5+
:--	---

PHP mysqli_options() 函数

实例

打开一个到 MySQL 服务器的新连接：

```
<?php
$con=mysqli_init();
if (!$con)
{
die("mysqli_init failed");
}

mysqli_options($con,MYSQLI_READ_DEFAULT_FILE,"myfile.cnf");

if (!mysqli_real_connect($con,"localhost","my_user","my_password","my_db"))
{
die("Connect Error: " . mysqli_connect_error());
}

mysqli_close($con);
?>
```

定义和用法

mysqli_options() 函数设置额外的连接选项，用于影响连接行为。

mysqli_options() 函数可以被调用若干次来设置若干个选项。

注释：mysqli_options() 函数可以在 mysqli_init() 之后和 mysqli_real_connect() 之前被调用。

语法

mysqli*options(_connection,option,value);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。
option	必需。规定要设置的选项。可以是下列值中的一个： `MYSQLI_OPT_CONNECT_TIMEOUT` - 以秒为单位的连接超时时间 `MYSQLI_OPT_LOCAL_INFILE` - 启用/禁用 LOAD LOCAL INFILE `MYSQLI_INIT_COMMAND` - 在连接到 MySQL 服务器之后的执行命令 `MYSQLI_READ_DEFAULT_FILE` - 从已命名的文件而不是 my.cnf 中读取选项 `MYSQLI_READ_DEFAULT_GROUP` - 从 my.cnf 或者 MYSQLI_READ_DEFAULT_FILE 中指定的文件中的已命名组中读取选项 `MYSQLI_SERVER_PUBLIC_KEY` - 基于 SHA-256 认证的 RSA 公共密钥文件
value	必需。规定 *option* 的值。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---
更新日志：	在 PHP 5.5 中新增了 MYSQLI_SERVER_PUBLIC_KEY 选项。
:--	---

PHP mysqli_ping() 函数

实例

进行一个服务器连接：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");

// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Check if server is alive
if (mysqli_ping($con))
{
echo "Connection is ok!";
}
else
{
echo "Error: ". mysqli_error($con);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_ping() 函数进行一个服务器连接，如果连接已断开则尝试重新连接。

语法

mysqli*ping(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_query() 函数

实例

执行针对数据库的查询：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Perform queries
mysqli_query($con,"SELECT * FROM Persons");
mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Glenn','Quagmire',33)");

mysqli_close($con);
?>
```

定义和用法

mysqli_query() 函数执行某个针对数据库的查询。

语法

mysqli*query(_connection,query,resultmode*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。
query	必需，规定查询字符串。
resultmode	可选。一个常量。可以是下列值中的任意一个： `MYSQLI_USE_RESULT` （如果需要检索大量数据，请使用这个） `MYSQLI_STORE_RESULT` （默认）

技术细节

返回值：	针对成功的 **SELECT、SHOW、DESCRIBE** 或 **EXPLAIN** 查询，将返回一个 **mysqli_result** 对象。针对其他成功的查询，将返回 **TRUE**。如果失败，则返回 **FALSE**。
PHP 版本：	5+
:--	---
更新日志：	在 PHP 5.3.0 中新增了异步查询的功能。
:--	---

PHP mysqli_real_connect() 函数

实例

打开一个到 MySQL 服务器的新连接：

```
<?php
$con=mysqli_init();
if (!$con)
{
die("mysqli_init failed");
}

if (!mysqli_real_connect($con,"localhost","my_user","my_password","my_db"))
{
die("Connect Error: " . mysqli_connect_error());
}

mysqli_close($con);
?>
```

定义和用法

mysqli_real_connect() 函数打开一个到 MySQL 服务器的新连接。

mysqli_real_connect() 函数与 mysqli_connect() 函数在以下几个方面存在差异：

- mysqli_real_connect() 要求一个由 mysqli_init() 创建的有效的对象。
- mysqli_real_connect() 可以与 mysqli_options() 一同使用来设置连接的不同选项。
- mysqli_real_connect() 有一个 flag 参数。

语法

mysqli*real_connect(_connection,host,username,password,dbname,port,socket,flag*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。
host	可选。规定主机名或 IP 地址。
username	可选。规定 MySQL 用户名。
password	可选。规定 MySQL 密码。
dbname	可选。规定要使用的默认数据库。
port	可选。规定尝试连接到 MySQL 服务器的端口号。
socket	可选。规定 socket 或要使用的已命名 pipe。
flag	可选。规定不同的连接选项。可能的值： `MYSQLI_CLIENT_COMPRESS` - 使用压缩协议 `MYSQLI_CLIENT_FOUND_ROWS` - 返回匹配的行数（不是受影响的行数） `MYSQLI_CLIENT_IGNORE_SPACE` - 在函数名后允许空格，使函数名保留字 `MYSQLI_CLIENT_INTERACTIVE` - 在关闭连接之前允许不活动的 interactive_timeout 秒 `MYSQLI_CLIENT_SSL` - 使用 SSL 加密

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_real_escape_string() 函数

实例

转义字符串中的特殊字符：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");

// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_query($con,"CREATE TABLE myPersons LIKE Persons");

$newpers="Da'Silva"

// This query will fail, cause we did not escape $newpers
mysqli_query($con,"INSERT into myPersons (Lastname) VALUES ('$newpers')");

$newpers=mysqli_real_escape_string($con,$newpers);

// This query will work, cause we escaped $newpers
mysqli_query($con,"INSERT into myPersons (Lastname) VALUES ('$newpers')");

mysqli_close($con);
?>
```

定义和用法

mysqli_real_escape_string() 函数转义在 SQL 语句中使用的字符串中的特殊字符。

语法

mysqli*real_escape_string(_connection,escapestring*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。
escapestring	必需。要转义的字符串。编码的字符是 NUL（ASCII 0）、\n、\r、\、'、" 和 Control-Z。

技术细节

返回值：	返回已转义的字符串。
PHP 版本：	5+
:--	---

PHP mysqli_refresh() 函数

定义和用法

mysqli_refresh() 函数刷新表或缓存，或者重置复制服务器信息。

语法

mysqli*refresh(_connection,options*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。
options	要刷新的选项。可以是下列中的一个或多个（用 OR 分隔）： `MYSQLI_REFRESH_GRANT` - 刷新授权表 `MYSQLI_REFRESH_LOG` - 刷新记录 `MYSQLI_REFRESH_TABLES` - 刷新表缓存 `MYSQLI_REFRESH_HOSTS` - 刷新主机缓存 `MYSQLI_REFRESH_STATUS` - 重置状态变量 `MYSQLI_REFRESH_THREADS` - 刷新线程缓存 `MYSQLI_REFRESH_SLAVE` - 重置主服务器信息，重启从服务器 `MYSQLI_REFRESH_MASTER` - 移除二进制日志索引中的二进制日志文件，并截断索引文件。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5.3+
:--	---

PHP mysqli_rollback() 函数

实例

关闭自动提交，做一些查询，提交查询，然后回滚当前事务：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Set autocommit to off
mysqli_autocommit($con,FALSE);

// Insert some values
mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Peter','Griffin',35)");
mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Glenn','Quagmire',33)");

// Commit transaction
mysqli_commit($con);

// Rollback transaction
mysqli_rollback($con);

// Close connection
mysqli_close($con);
?>
```

定义和用法

mysqli_rollback() 函数回滚指定数据库连接的当前事务。

提示：请查看 mysqli_commit() 函数，用于提交指定数据库连接的当前事务。请查看 mysqli_autocommit() 函数，用于开启或关闭自动提交数据库修改。

语法

mysqli*rollback(_connection);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_select_db() 函数

实例

更改连接的默认数据库：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// ...some PHP code for database "my_db"...

// Change database to "test"
mysqli_select_db($con,"test");

// ...some PHP code for database "test"...

mysqli_close($con);
?>
```

定义和用法

mysqli_select_db() 函数用于更改连接的默认数据库。

语法

mysqli*select_db(_connection,dbname);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。
dbname	必需，规定要使用的默认数据库。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_set_charset() 函数

实例

设置默认客户端字符集：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Change character set to utf8
mysqli_set_charset($con,"utf8");

mysqli_close($con);
?>
```

定义和用法

mysqli_set_charset() 函数规定当与数据库服务器进行数据传送时要使用的默认字符集。

注释：在 Windows 平台上使用该函数，您需要 MySQL 客户端库 4.1.11 或以上版本（MySQL 5.0 需要 5.0.6 或以上版本）。

语法

mysqli*set_charset(_connection,charset);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。
charset	必需。规定默认字符集。

技术细节

返回值：	如果成功则返回 **TRUE**，如果失败则返回 **FALSE**。
PHP 版本：	5.0.5+
:--	---

PHP mysqli_sqlstate() 函数

实例

返回最后一个 MySQL 操作的 SQLSTATE 错误代码：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Table Persons already exists, so we should get an error
$sql="CREATE TABLE Persons (Firstname VARCHAR(30),Lastname VARCHAR(30),Age INT)"
if (!mysqli_query($con,$sql))
{
echo "SQLSTATE error: ". mysqli_sqlstate($con);
}

// Close connection
mysqli_close($con);
?>
```

定义和用法

mysqli_sqlstate() 函数返回最后一个错误的 SQLSTATE 错误代码。

错误代码包含五个字符。"00000" 表明没有错误。值由 ANSI SQL 和 ODBC 指定。

语法

mysqli*sqlstate(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个包含最后一个错误的 **SQLSTATE** 错误代码的字符串。
PHP 版本：	5+
:--	---

PHP mysqli_ssl_set() 函数

实例

创建 SSL 连接：

```
<?php
$con=mysqli_init();
if (!$con)
{
die("mysqli_init failed");
}

mysqli_ssl_set($con,"key.pem","cert.pem","cacert.pem",NULL,NULL);

if (!mysqli_real_connect($con,"localhost","my_user","my_password","my_db"))
{
die("Connect Error: " . mysqli_connect_error());
}

// Some queries...

mysqli_close($con);
?>
```

定义和用法

mysqli_ssl_set() 函数用于创建 SSL 安全连接。然而，该函数只有在启用 OpenSSL 支持时才有效。

注释：该函数必须在 mysqli_real_connect() 之前调用。

注释：在 PHP 5.3.3 之前的版本，MySQL Native Driver 不支持 SSL。自 PHP 5.3+ 起，在微软 Windows 上默认启用 MySQL Native Driver。

语法

mysqli*ssl_set(_connection,key,cert,ca,capath,cipher);*

参数	描述
connection	必需。规定要使用的 MySQL 连接。
key	必需。规定密钥文件的路径名。
cert	必需。规定认证文件的路径名。
ca	必需。规定认证授权文件的路径名。
capath	必需。规定包含 PEM 格式的可信 SSL CA 认证的目录的路径名。
cipher	必需。规定用于 SSL 加密的可用密码列表。

技术细节

返回值：	总是返回 **TRUE**。如果 **SSL** 安装不正确，当您尝试连接的时候，**mysqli_real_connect()** 将返回一个错误。
PHP 版本：	5+
:--	---

PHP mysqli_stat() 函数

实例

创建 SSL 连接：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

echo "System status: ". mysqli_stat($con);

mysqli_close($con);
?>
```

定义和用法

mysqli_stat() 函数返回当前系统状态。

语法

mysqli*stat(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个描述服务器状态的字符串。如果发生错误则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP mysqli_stmt_init() 函数

实例

初始化声明并返回 mysqli_stmt_prepare() 使用的对象：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$city="Sandnes";

// Create a prepared statement
$stmt=mysqli_stmt_init($con);

if (mysqli_stmt_prepare($stmt,"SELECT District FROM City WHERE Name=?"))
{

// Bind parameters
mysqli_stmt_bind_param($stmt,"s",$city);

// Execute query
mysqli_stmt_execute($stmt);

// Bind result variables
mysqli_stmt_bind_result($stmt,$district);

// Fetch value
mysqli_stmt_fetch($stmt);

printf("%s is in district %s",$city,$district);

// Close statement
mysqli_stmt_close($stmt);
}

mysqli_close($con);
?>
```

定义和用法

mysqli_stmt_init() 函数初始化声明并返回 mysqli_stmt_prepare() 使用的对象。

语法

mysqli*stmt_init(_connection*)*;*

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回一个对象。
PHP 版本：	5+
:--	---

PHP mysqli_thread_id() 函数

实例

返回当前连接的线程 ID，然后杀死连接：

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno($con))
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Get thread id
$t_id=mysqli_thread_id($con);

// Kill connection
mysqli_kill($con,$t_id);
?>
```

定义和用法

mysqli_thread_id() 函数返回当前连接的线程 ID，然后使用 mysqli_kill() 函数杀死该连接。

注释：如果连接被损坏且重新连接，线程 ID 将会改变。因此，仅当您需要的时候才获取线程 ID。

语法

mysqli*thread_id(_connection*);

参数	描述
connection	必需。规定要使用的 MySQL 连接。

技术细节

返回值：	返回当前连接的线程 **ID**。
PHP 版本：	5+
:--	---

PHP mysqli_thread_safe() 函数

定义和用法

mysqli_thread_safe() 函数返回是否将客户端库编译成 thread-safe。

语法

mysqli*thread_safe()*;_

技术细节

返回值：	如果客户端库是 **thread-safe** 则返回 **TRUE**，否则返回 **FALSE**。
PHP 版本：	5+
:--	---

PHP PDO

PHP 数据对象 （PDO） 扩展为PHP访问数据库定义了一个轻量级的一致接口。

PDO 提供了一个数据访问抽象层，这意味着，不管使用哪种数据库，都可以用相同的函数（方法）来查询和获取数据。

PDO随PHP5.1发行，在PHP5.0的PECL扩展中也可以使用，无法运行于之前的PHP版本。

PDO 安装

你可以通过 PHP 的 phpinfo() 函数来查看是否安装了PDO扩展。

在 Unix 系统上安装 PDO

在Unix上或Linux上你需要添加以下扩展：

```
extension=pdo.so
```

Windows 用户

PDO 和所有主要的驱动作为共享扩展随 PHP 一起发布，要激活它们只需简单地编辑 php.ini 文件，并添加以下扩展：

```
extension=php_pdo.dll
```

除此之外还有以下对应的各种数据库扩展：

```
;extension=php_pdo_firebird.dll
;extension=php_pdo_informix.dll
;extension=php_pdo_mssql.dll
;extension=php_pdo_mysql.dll
;extension=php_pdo_oci.dll
;extension=php_pdo_oci8.dll
;extension=php_pdo_odbc.dll
;extension=php_pdo_pgsql.dll
;extension=php_pdo_sqlite.dll
```

在设定好这些配置后，我们需要重启PHP 或 Web服务器。

接下来我们们来看下具体的实例，以下为使用PDO连接MySql数据库的实例：

```
<?php
$dbms='mysql';     //数据库类型
$host='localhost'; //数据库主机名
$dbName='test';    //使用的数据库
$user='root';      //数据库连接用户名
$pass='';          //对应的密码
$dsn="$dbms:host=$host;dbname=$dbName";

try {
    $dbh = new PDO($dsn, $user, $pass); //初始化一个PDO对象
    echo "连接成功<br/>";
    /*你还可以进行一次搜索操作
    foreach ($dbh->query('SELECT * from FOO') as $row) {
        print_r($row); //你可以用 echo($GLOBAL); 来看到这些值
    }
    */
    $dbh = null;
} catch (PDOException $e) {
    die ("Error!: " . $e->getMessage() . "<br/>");
}
//默认这个不是长连接，如果需要数据库长连接，需要最后加一个参数：array(PDO::ATTR_PERSISTENT => true)
$db = new PDO($dsn, $user, $pass, array(PDO::ATTR_PERSISTENT => true));

?>
```

很简单吧，接下来就让我们来具体看下PHP PDO具体说明：

- 预定义常量
- PDO 类：
    - PDO::beginTransaction — 启动一个事务
    - PDO::commit — 提交一个事务
    - PDO::__construct — 创建一个表示数据库连接的 PDO 实例
    - PDO::errorCode — 获取跟数据库句柄上一次操作相关的 SQLSTATE
    - PDO::errorInfo — 返回最后一次操作数据库的错误信息
    - PDO::exec — 执行一条 SQL 语句，并返回受影响的行数
    - PDO::getAttribute — 取回一个数据库连接的属性
    - PDO::getAvailableDrivers — 返回一个可用驱动的数组
    - PDO::inTransaction — 检查是否在一个事务内
    - PDO::lastInsertId — 返回最后插入行的ID或序列值
    - PDO::prepare — 备要执行的SQL语句并返回一个 PDOStatement 对象
    - PDO::query — 执行 SQL 语句，返回PDOStatement对象,可以理解为结果集
    - PDO::quote — 为SQL语句中的字符串添加引号。
    - PDO::rollBack — 回滚一个事务
    - PDO::setAttribute — 设置属性
- PDOStatement 类：
    - PDOStatement::bindColumn — 绑定一列到一个 PHP 变量
    - PDOStatement::bindParam — 绑定一个参数到指定的变量名
    - PDOStatement::bindValue — 把一个值绑定到一个参数

- PDOStatement::closeCursor — 关闭游标，使语句能再次被执行。
- PDOStatement::columnCount — 返回结果集中的列数
- PDOStatement::debugDumpParams — 打印一条 SQL 预处理命令
- PDOStatement::errorCode — 获取跟上一次语句句柄操作相关的 SQLSTATE
- PDOStatement::errorInfo — 获取跟上一次语句句柄操作相关的扩展错误信息
- PDOStatement::execute — 执行一条预处理语句
- PDOStatement::fetch — 从结果集中获取下一行
- PDOStatement::fetchAll — 返回一个包含结果集中所有行的数组
- PDOStatement::fetchColumn — 从结果集中的下一行返回单独的一列。
- PDOStatement::fetchObject — 获取下一行并作为一个对象返回。
- PDOStatement::getAttribute — 检索一个语句属性
- PDOStatement::getColumnMeta — 返回结果集中一列的元数据
- PDOStatement::nextRowset — 在一个多行集语句句柄中推进到下一个行集
- PDOStatement::rowCount — 返回受上一个 SQL 语句影响的行数
- PDOStatement::setAttribute — 设置一个语句属性
- PDOStatement::setFetchMode — 为语句设置默认的获取模式。

PHP PDO预定义常量

以下常量由本扩展模块定义，因此只有在本扩展的模块被编译到PHP中，或者在运行时被动态加载后才有效。

注意： PDO使用类常量自PHP 5.1。以前的版本使用的全局常量形式PDO_PARAM_BOOL中。

PDO::PARAM_BOOL (integer)	
PDO::PARAM_NULL (integer)	表示 SQL 中的 NULL 数据类型。
PDO::PARAM_INT (integer)	表示 SQL 中的整型。
PDO::PARAM_STR (integer)	表示 SQL 中的 CHAR、VARCHAR 或其他字
PDO::PARAM_LOB (integer)	表示 SQL 中大对象数据类型。
PDO::PARAM_STMT (integer)	表示一个记录集类型。当前尚未被任何驱动支
PDO::PARAM_INPUT_OUTPUT (integer)	指定参数为一个存储过程的 INOUT 参数。必
PDO::FETCH_LAZY (integer)	指定获取方式，将结果集中的每一行作为一个 名。在 PDOStatement::fetchAll() 中无效。
PDO::FETCH_ASSOC (integer)	指定获取方式，将对应结果集中的每一行作为 PDO::FETCH_ASSOC每个列名只返回一个值
PDO::FETCH_NAMED (integer)	指定获取方式，将对应结果集中的每一行作为 PDO::FETCH_ASSOC每个列名 返回一个包含
PDO::FETCH_NUM (integer)	指定获取方式，将对应结果集中的每一行作为
PDO::FETCH_BOTH (integer)	指定获取方式，将对应结果集中的每一行作为
PDO::FETCH_OBJ (integer)	指定获取方式，将结果集中的每一行作为一个
PDO::FETCH_BOUND (integer)	指定获取方式，返回 TRUE 且将结果集中的列 PHP 变量。
PDO::FETCH_COLUMN (integer)	指定获取方式，从结果集中的下一行返回所需
PDO::FETCH_CLASS (integer)	指定获取方式，返回一个所请求类的新实例， 方法
PDO::FETCH_INTO (integer)	指定获取方式，更新一个请求类的现有实例，
PDO::FETCH_FUNC (integer)	允许在运行中完全用自定义的方式处理数据。
PDO::FETCH_GROUP (integer)	根据值分组返回。通常和 PDO::FETCH_COL
PDO::FETCH_UNIQUE (integer)	只取唯一值。
PDO::FETCH_KEY_PAIR (integer)	获取一个有两列的结果集到一个数组，其中第
PDO::FETCH_CLASSTYPE (integer)	根据第一列的值确定类名。

PDO::FETCH_SERIALIZE (integer)	类似 PDO::FETCH_INTO， 但是以一个序列化 构造函数从不会被调用。
PDO::FETCH_PROPS_LATE (integer)	设置属性前调用构造函数。自 PHP 5.2.0 起可
PDO::ATTR_AUTOCOMMIT (integer)	如果此值为 FALSE， PDO 将试图禁用自动提
PDO::ATTR_PREFETCH (integer)	设置预取大小来为你的应用平衡速度和内存使 时也会占用更多的内存。
PDO::ATTR_TIMEOUT (integer)	设置连接数据库的超时秒数。
PDO::ATTR_ERRMODE (integer)	关于此属性的更多信息请参见 错误及错误处理
PDO::ATTR_SERVER_VERSION (integer)	此为只读属性；返回 PDO 所连接的数据库服
PDO::ATTR_CLIENT_VERSION (integer)	此为只读属性；返回 PDO 驱动所用客户端库
PDO::ATTR_SERVER_INFO (integer)	此为只读属性。返回一些关于 PDO 所连接的
PDO::ATTR_CONNECTION_STATUS (integer)	
PDO::ATTR_CASE (integer)	用类似 PDO::CASE_* 的常量强制列名为指定
PDO::ATTR_CURSOR_NAME (integer)	获取或设置使用游标的名称。当使用可滚动游
PDO::ATTR_CURSOR (integer)	选择游标类型。 PDO 当前支持 PDO::CURSO 确实需要一个可滚动游标。
PDO::ATTR_DRIVER_NAME (string)	返回驱动名称。使用 PDO::ATTR_DRIVER_N 子： if ($db->getAttribute(PDO::ATTR_DRIV
PDO::ATTR_ORACLE_NULLS (integer)	在获取数据时将空字符串转换成 SQL 中的 NU
PDO::ATTR_PERSISTENT (integer)	请求一个持久连接，而非创建一个新连接。关
PDO::ATTR_STATEMENT_CLASS (integer)	
PDO::ATTR_FETCH_CATALOG_NAMES (integer)	将包含的目录名添加到结果集中的每个列名前 支持此属性。
PDO::ATTR_FETCH_TABLE_NAMES (integer)	将包含的表名添加到结果集中的每个列名前面 此属性。
PDO::ATTR_STRINGIFY_FETCHES (integer)	
PDO::ATTR_MAX_COLUMN_LEN (integer)	
PDO::ATTR_DEFAULT_FETCH_MODE (integer)	自 PHP 5.2.0 起可用。
PDO::ATTR_EMULATE_PREPARES (integer)	自 PHP 5.1.3 起可用。
PDO::ERRMODE_SILENT (integer)	如果发生错误，则不显示错误或异常。希望开

PDO::ERRMODE_WARNING (integer)	如果发生错误，则显示一个 PHP E_WARNIN
PDO::ERRMODE_EXCEPTION (integer)	如果发生错误，则抛出一个 PDOException 异
PDO::CASE_NATURAL (integer)	保留数据库驱动返回的列名。
PDO::CASE_LOWER (integer)	强制列名小写。
PDO::CASE_UPPER (integer)	强制列名大写。
PDO::NULL_NATURAL (integer)	
PDO::NULL_EMPTY_STRING (integer)	
PDO::NULL_TO_STRING (integer)	
PDO::FETCH_ORI_NEXT (integer)	在结果集中获取下一行。仅对可滚动游标有效
PDO::FETCH_ORI_PRIOR (integer)	在结果集中获取上一行。仅对可滚动游标有效
PDO::FETCH_ORI_FIRST (integer)	在结果集中获取第一行。仅对可滚动游标有效
PDO::FETCH_ORI_LAST (integer)	在结果集中获取最后一行。仅对可滚动游标有
PDO::FETCH_ORI_ABS (integer)	根据行号从结果集中获取需要的行。仅对可滚
PDO::FETCH_ORI_REL (integer)	根据当前游标位置的相对位置从结果集中获取
PDO::CURSOR_FWDONLY (integer)	创建一个只进游标的 PDOStatement 对象。此
PDO::CURSOR_SCROLL (integer)	创建一个可滚动游标的 PDOStatement 对象。
PDO::ERR_NONE (string)	对应 SQLSTATE '00000'，表示 SQL 语句没有 否有错误发生时，此常量非常方便。在检查上
PDO::PARAM_EVT_ALLOC (integer)	分配事件
PDO::PARAM_EVT_FREE (integer)	解除分配事件
PDO::PARAM_EVT_EXEC_PRE (integer)	执行一条预处理语句之前触发事件。
PDO::PARAM_EVT_EXEC_POST (integer)	执行一条预处理语句之后触发事件。
PDO::PARAM_EVT_FETCH_PRE (integer)	从一个结果集中取出一条结果之前触发事件。
PDO::PARAM_EVT_FETCH_POST (integer)	从一个结果集中取出一条结果之后触发事件。
PDO::PARAM_EVT_NORMALIZE (integer)	在绑定参数注册允许驱动程序正常化变量名时

PHP PDO连接

连接是通过创建 PDO 基类的实例而建立的。不管使用哪种驱动程序，都是用 PDO 类名。

连接到 MySQL

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
?>
```

注意：如果有任何连接错误，将抛出一个 PDOException 异常对象。

处理连接错误

```
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
    foreach($dbh->query('SELECT * from FOO') as $row) {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

连接数据成功后，返回一个 PDO 类的实例给脚本，此连接在 PDO 对象的生存周期中保持活动。

要想关闭连接，需要销毁对象以确保所有剩余到它的引用都被删除，可以赋一个 NULL 值给对象变量。

如果不这么做，PHP 在脚本结束时会自动关闭连接。

关闭一个连接:

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
// 在此使用连接

// 现在运行完成，在此关闭连接
$dbh = null;
?>
```

很多 web 应用程序通过使用到数据库服务的持久连接获得好处。

持久连接在脚本结束后不会被关闭，且被缓存，当另一个使用相同凭证的脚本连接请求时被重用。

持久连接缓存可以避免每次脚本需要与数据库回话时建立一个新连接的开销，从而让 web 应用程序更快。

持久化连接

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass, array(
    PDO::ATTR_PERSISTENT => true
));
?>
```

注意：如果想使用持久连接，必须在传递给 PDO 构造函数的驱动选项数组中设置 PDO::ATTR_PERSISTENT 。如果是在对象初始化之后用 PDO::setAttribute() 设置此属性，则驱动程序将不会使用持久连接。

PHP PDO 事务与自动提交

现在通过 PDO 连接上了，在开始进行查询前，必须先理解 PDO 是如何管理事务的。

事务支持四大特性（ACID）：

- 原子性（Atomicity）
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability）

通俗地讲，在一个事务中执行的任何操作，即使是分阶段执行的，也能保证安全地应用于数据库，并在提交时不会受到来自其他连接的干扰。

事务操作也可以根据请求自动撤销（假设还没有提交），这使得在脚本中处理错误更加容易。

事务通常是通过把一批更改"积蓄"起来然后使之同时生效而实现的；这样做的好处是可以大大地提供这些更改的效率。

换句话说，事务可以使脚本更快，而且可能更健壮（不过需要正确地使用事务才能获得这样的好处）。

不幸的是，并非每种数据库都支持事务，因此当第一次打开连接时，PDO 需要在所谓的"自动提交"模式下运行。

自动提交模式意味着，如果数据库支持，运行的每个查询都有它自己的隐式事务，如果数据库不支持事务，则没有。

如果需要一个事务，则必须用 PDO::beginTransaction() 方法来启动。如果底层驱动不支持事务，则抛出一个 PDOException 异常（不管错误处理设置是怎样的，这都是一个严重的错误状态）。

一旦开始了事务，可用 PDO::commit() 或 PDO::rollBack()来完成，这取决于事务中的代码是否运行成功。

注意：PDO 仅在驱动层检查是否具有事务处理能力。如果某些运行时条件意味着事务不可用，且数据库服务接受请求去启动一个事务，PDO::beginTransaction() 将仍然返回 TRUE 而且没有错误。试着在 MySQL 数据库的 MyISAM 数据表中使用事务就是一个很好的例子。

当脚本结束或连接即将被关闭时，如果尚有一个未完成的事务，那么 PDO 将自动回滚该事务。这种安全措施有助于在脚本意外终止时避免出现不一致的情况——如果没有显式地提交事务，那么假设是某个地方出错了，所以执行回滚来保证数据安全。

注意： 只有通过 PDO::beginTransaction() 启动一个事务后，才可能发生自动回滚。如果手动发出一条查询启动事务， 则 PDO 无法知晓，从而在必要时不能进行回滚。

在事务中执行批处理:

在下面例子中，假设为新员工创建一组条目，分配一个为23的ID。除了登记此人的基本数据之外，还需要记录他的工资。

两个更新分别完成起来很简单，但通过封闭在 PDO::beginTransaction() 和PDO::commit() 调用中，可以保证在更改完成之前，其他人无法看到这些更改。

如果发生了错误，catch 块回滚自事务启动以来发生的所有更改，并输出一条错误信息。

```
<?php
try {
  $dbh = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2',
      array(PDO::ATTR_PERSISTENT => true));
  echo "Connected\n";
} catch (Exception $e) {
  die("Unable to connect: " . $e->getMessage());
}

try {
  $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $dbh->beginTransaction();
  $dbh->exec("insert into staff (id, first, last) values (23, 'Joe', 'Bloggs')");
  $dbh->exec("insert into salarychange (id, amount, changedate)
      values (23, 50000, NOW())");
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

并不局限于在事务中更改，也可以发出复杂的查询来提取数据，还可以使用那些信息来构建更多的更改和查询；当事务激活时，可以保证其他人在操作进行当中无法作出更改。

PHP PDO 预处理语句与存储过程

很多更成熟的数据库都支持预处理语句的概念。

什么是预处理语句？可以把它看作是想要运行的 SQL 的一种编译过的模板，它可以使用变量参数进行定制。预处理语句可以带来两大好处：

- 查询仅需解析（或预处理）一次，但可以用相同或不同的参数执行多次。当查询准备好后，数据库将分析、编译和优化执行该查询的计划。对于复杂的查询，此过程要花费较长的时间，如果需要以不同参数多次重复相同的查询，那么该过程将大大降低应用程序的速度。通过使用预处理语句，可以避免重复分析/编译/优化周期。简言之，预处理语句占用更少的资源，因而运行得更快。
- 提供给预处理语句的参数不需要用引号括起来，驱动程序会自动处理。如果应用程序只使用预处理语句，可以确保不会发生SQL 注入。（然而，如果查询的其他部分是由未转义的输入来构建的，则仍存在 SQL 注入的风险）。

预处理语句如此有用，以至于它们唯一的特性是在驱动程序不支持的时PDO 将模拟处理。这样可以确保不管数据库是否具有这样的功能，都可以确保应用程序可以用相同的数据访问模式。

用预处理语句进行重复插入

下面例子通过用 name 和 value 替代相应的命名占位符来执行一个插入查询

```
<?php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// 插入一行
$name = 'one';
$value = 1;
$stmt->execute();

//  用不同的值插入另一行
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

用预处理语句进行重复插入

下面例子通过用 name 和 value 取代 ? 占位符的位置来执行一条插入查询。

```
<?php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (?, ?)");
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $value);

// 插入一行
$name = 'one';
$value = 1;
$stmt->execute();

// 用不同的值插入另一行
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

使用预处理语句获取数据

下面例子获取数据基于键值已提供的形式。用户的输入被自动用引号括起来，因此不会有 SQL 注入攻击的危险。

```
<?php
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");
if ($stmt->execute(array($_GET['name']))) {
  while ($row = $stmt->fetch()) {
    print_r($row);
  }
}
?>
```

如果数据库驱动支持，应用程序还可以绑定输出和输入参数.输出参数通常用于从存储过程获取值。输出参数使用起来比输入参数要稍微复杂一些，因为当绑定一个输出参数时，必须知道给定参数的长度。如果为参数绑定的值大于建议的长度，就会产生一个错误。

带输出参数调用存储过程

```
<?php
$stmt = $dbh->prepare("CALL sp_returns_string(?)");
$stmt->bindParam(1, $return_value, PDO::PARAM_STR, 4000);

// 调用存储过程
$stmt->execute();

print "procedure returned $return_value\n";
?>
```

还可以指定同时具有输入和输出值的参数，其语法类似于输出参数。在下一个例子中，字符串"hello"被传递给存储过程，当存储过程返回时，hello 被替换为该存储过程返回的值。

带输入/输出参数调用存储过程

```
<?php
$stmt = $dbh->prepare("CALL sp_takes_string_returns_string(?)");
$value = 'hello';
$stmt->bindParam(1, $value, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 4000);

// 调用存储过程
$stmt->execute();

print "procedure returned $value\n";
?>
```

占位符的无效使用

```
<?php
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name LIKE '%?%'");
$stmt->execute(array($_GET['name']));

// 占位符必须被用在整个值的位置
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name LIKE ?");
$stmt->execute(array("%$_GET[name]%"));
?>
```

PHP PDO 错误与错误处理

- **PDO::ERRMODE_SILENT**

  此为默认模式。 PDO 将只简单地设置错误码，可使用 PDO::errorCode() 和 PDO::errorInfo() 方法来检查语句和数据库对象。如果错误是由于对语句对象的调用而产生的，那么可以调用那个对象的 PDOStatement::errorCode() 或 PDOStatement::errorInfo() 方法。如果错误是由于调用数据库对象而产生的，那么可以在数据库对象上调用上述两个方法。

- **PDO::ERRMODE_WARNING**

  除设置错误码之外，PDO 还将发出一条传统的 E_WARNING 信息。如果只是想看看发生了什么问题且不中断应用程序的流程，那么此设置在调试/测试期间非常有用。

- **PDO::ERRMODE_EXCEPTION**

  除设置错误码之外，PDO 还将抛出一个 PDOException 异常类并设置它的属性来反射错误码和错误信息。此设置在调试期间也非常有用，因为它会有效地放大脚本中产生错误的点，从而可以非常快速地指出代码中有问题的潜在区域（记住：如果异常导致脚本终止，则事务被自动回滚）。

  异常模式另一个非常有用的是，相比传统 PHP 风格的警告，可以更清晰地构建自己的错误处理，而且比起静默模式和显式地检查每种数据库调用的返回值，异常模式需要的代码/嵌套更少。

创建 PDO 实例并设置错误模式

```
<?php
$dsn = 'mysql:dbname=testdb;host=127.0.0.1';
$user = 'dbuser';
$password = 'dbpass';

try {
    $dbh = new PDO($dsn, $user, $password);
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}

?>
```

注意： 不管当前是否设置了 PDO::ATTR_ERRMODE ，如果连接失败，PDO::__construct() 将总是抛出一个 PDOException 异常。未捕获异常是致命的。

创建 PDO 实例并在构造函数中设置错误模式

```
<?php
$dsn = 'mysql:dbname=test;host=127.0.0.1';
$user = 'googleguy';
$password = 'googleguy';

/*
    使用 try/catch 围绕构造函数仍然有效，即使设置了 ERRMODE 为 WARNING，
    因为如果连接失败，PDO::__construct 将总是抛出一个  PDOException 异常。
*/
try {
    $dbh = new PDO($dsn, $user, $password, array(PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNIN
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
    exit;
}

// 这里将导致 PDO 抛出一个 E_WARNING 级别的错误，而不是 一个异常 （当数据表不存在时）
$dbh->query("SELECT wrongcolumn FROM wrongtable");
?>
```

以上例程会输出：

```
Warning: PDO::query(): SQLSTATE[42S02]: Base table or view not found: 1146 Table 'test.wr
/tmp/pdo_test.php on line 18
add a note add a note
```

PHP PDO 大对象 (LOBs)

应用程序在某一时刻，可能需要在数据库中存储"大"数据。

"大"通常意味着"大约 4kb 或以上"，尽管某些数据库在数据达到"大"之前可以轻松地处理多达 32kb 的数据。大对象本质上可能是文本或二进制。

在 PDOStatement::bindParam() 或 PDOStatement::bindColumn()) 调用中使用 PDO::PARAM_LOB 类型码可以让 PDO 使用大数据类型。

PDO::PARAM_LOB 告诉 PDO 作为流来映射数据，以便能使用 PHP Streams API 来操作。

从数据库中显示一张图片

下面例子绑定一个 LOB 到 $lob 变量，然后用 fpassthru() 将其发送到浏览器。因为 LOB 代表一个流，所以类似 fgets()、fread() 以及 stream_get_contents() 这样的函数都可以用在它上面。

```
<?php
$db = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2');
$stmt = $db->prepare("select contenttype, imagedata from images where id=?");
$stmt->execute(array($_GET['id']));
$stmt->bindColumn(1, $type, PDO::PARAM_STR, 256);
$stmt->bindColumn(2, $lob, PDO::PARAM_LOB);
$stmt->fetch(PDO::FETCH_BOUND);

header("Content-Type: $type");
fpassthru($lob);
?>
```

插入一张图片到数据库

下面例子打开一个文件并将文件句柄传给 PDO 来做为一个 LOB 插入。PDO尽可能地让数据库以最有效的方式获取文件内容。

```
<?php
$db = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2');
$stmt = $db->prepare("insert into images (id, contenttype, imagedata) values (?, ?, ?)");
$id = get_new_id(); // 调用某个函数来分配一个新 ID

// 假设处理一个文件上传
// 可以在 PHP 文档中找到更多的信息

$fp = fopen($_FILES['file']['tmp_name'], 'rb');

$stmt->bindParam(1, $id);
$stmt->bindParam(2, $_FILES['file']['type']);
$stmt->bindParam(3, $fp, PDO::PARAM_LOB);

$db->beginTransaction();
$stmt->execute();
$db->commit();
?>
```

插入一张图片到数据库：Oracle

对于从文件插入一个 lob，Oracle略有不同。必须在事务之后进行插入，否则当执行查询时导致新近插入 LOB 将以0长度被隐式提交：

```
<?php
$db = new PDO('oci:', 'scott', 'tiger');
$stmt = $db->prepare("insert into images (id, contenttype, imagedata) " .
"VALUES (?, ?, EMPTY_BLOB()) RETURNING imagedata INTO ?");
$id = get_new_id(); // 调用某个函数来分配一个新 ID

// 假设处理一个文件上传
// 可以在 PHP 文档中找到更多的信息

$fp = fopen($_FILES['file']['tmp_name'], 'rb');

$stmt->bindParam(1, $id);
$stmt->bindParam(2, $_FILES['file']['type']);
$stmt->bindParam(3, $fp, PDO::PARAM_LOB);

$stmt->beginTransaction();
$stmt->execute();
$stmt->commit();
?>
```

PDO::beginTransaction

PDO::beginTransaction 启动一个事务(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
bool PDO::beginTransaction ( void )
```

关闭自动提交模式。自动提交模式被关闭的同时，通过 PDO 对象实例对数据库做出的更改直到调用 PDO::commit() 结束事务才被提交。

调用 PDO::rollBack() 将回滚对数据库做出的更改并将数据库连接返回到自动提交模式。

包括 MySQL 在内的一些数据库，当发出一条类似 DROP TABLE 或 CREATE TABLE 这样的 DDL 语句时，会自动进行一个隐式地事务提交。

隐式地提交将阻止你在此事务范围内回滚任何其他更改。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

回滚一个事务

下面例子在回滚此更改前开始一个事务并发出两条修改数据库的语句。

但在 MySQL 中，DROP TABLE 语句自动提交事务，使得在此事务中的任何更改都不会被回滚。

```
<?php
/* 开始一个事务，关闭自动提交 */
$dbh->beginTransaction();

/*  更改数据库架构及数据 */
$sth = $dbh->exec("DROP TABLE fruit");
$sth = $dbh->exec("UPDATE dessert
    SET name = 'hamburger'");

/*  识别出错误并回滚更改 */
$dbh->rollBack();

/* 数据库连接现在返回到自动提交模式 */
?>
```

PDO::commit

PDO::commit提交一个事务(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
bool PDO::commit ( void )
```

提交一个事务，数据库连接返回到自动提交模式直到下次调用 PDO::beginTransaction() 开始一个新的事务为止。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

提交一个基础事务

```
<?php
/* 开始一个事务，关闭自动提交 */
$dbh->beginTransaction();

/* 在全有或全无的基础上插入多行记录（要么全部插入，要么全部不插入） */
$sql = 'INSERT INTO fruit
    (name, colour, calories)
    VALUES (?, ?, ?)';

$sth = $dbh->prepare($sql);

foreach ($fruits as $fruit) {
    $sth->execute(array(
        $fruit->name,
        $fruit->colour,
        $fruit->calories,
    ));
}

/* 提交更改 */
$dbh->commit();

/* 现在数据库连接返回到自动提交模式 */
?>
```

提交一个DDL事务

```
<?php
/*  开始一个事务，关闭自动提交 */
$dbh->beginTransaction();

/* Change the database schema */
$sth = $dbh->exec("DROP TABLE fruit");

/* 更改数据库架构 */
$dbh->commit();

/* 现在数据库连接返回到自动提交模式 */
?>
```

注意：并不是所有数据库都允许使用DDL语句进行事务操作：有些会产生错误，而其他一些（包括MySQL）会在遇到第一个DDL语句后就自动提交事务。

PDO::__construct

PDO::__construct — 创建一个表示数据库连接的 PDO 实例(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
PDO::__construct ( string $dsn [, string $username [, string $password [, array $driver_o
```

创建一个表示连接到请求数据库的数据库连接 PDO 实例。

参数说明

- **dsn**：数据源名称或叫做 DSN，包含了请求连接到数据库的信息。
- **username**： DSN字符串中的用户名。对于某些PDO驱动，此参数为可选项。
- **password**： DSN字符串中的密码。对于某些PDO驱动，此参数为可选项。
- **driver_options**： 一个具体驱动的连接选项的键=>值数组。

返回值

成功则返回一个PDO对象。

错误／异常

如果试图连接到请求的数据库失败，则PDO::__construct() 抛出一个 PDO异常（PDOException）。

实例

通过调用驱动程序创建一个**PDO**实例

```
<?php
/* 通过调用驱动程序创建一个PDO实例 */
$dsn = 'mysql:dbname=testdb;host=127.0.0.1';
$user = 'dbuser';
$password = 'dbpass';

try {
    $dbh = new PDO($dsn, $user, $password);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}

?>
```

PDO::errorCode

PDO::errorCode — 获取跟数据库句柄上一次操作相关的 SQLSTATE(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
mixed PDO::errorCode ( void )
```

返回值

返回一个 SQLSTATE，一个由5个字母或数字组成的在 ANSI SQL 标准中定义的标识符。 简要地说，一个 SQLSTATE 由前面两个字符的类值和后面三个字符的子类值组成。

如果数据库句柄没有进行操作，则返回 NULL 。

实例

取得一个 SQLSTATE 码

```
/* 引发一个错误 -- BONES 数据表不存在 */
$dbh->exec("INSERT INTO bones(skull) VALUES ('lucy')");

echo "\nPDO::errorCode(): ";
print $dbh->errorCode();
?>
```

以上例程会输出：

```
PDO::errorCode(): 42S02
```

PDO::errorInfo

PDO::errorCode — 返回最后一次操作数据库的错误信息(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
public array PDO::errorInfo ( void )
```

返回值

返回一个数组，该数组包含了最后一次操作数据库的错误信息描述。

数组内容如下：

元素	信息
0	SQLSTATE 错误码 (5个字母或数字组成的在 ANSI SQL 标准中定义的标识符).
1	错误代码
2	错误信息

注意：如果数据库句柄没有进行操作，则返回 NULL 。

实例

显示errorInfo()中关于PDO_ODBC连接到DB2数据库的错误信息

```
<?php
/* 错误的SQL语法 */
$stmt = $dbh->prepare('bogus sql');
if (!$stmt) {
    echo "\nPDO::errorInfo():\n";
    print_r($dbh->errorInfo());
}
?>
```

以上例程会输出：

```
PDO::errorInfo():
Array
(
    [0] => HY000
    [1] => 1
    [2] => near "bogus": syntax error
)
```

PDO::exec

PDO::exec — 执行一条 SQL 语句，并返回受影响的行数(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
int PDO::exec ( string $statement )
```

PDO::exec() 在一个单独的函数调用中执行一条 SQL 语句，返回受此语句影响的行数。

PDO::exec() 不会从一条 SELECT 语句中返回结果。对于在程序中只需要发出一次的 SELECT 语句，可以考虑使用 PDO::query()。

参数说明：

statement： 要被预处理和执行的 SQL 语句。

返回值

PDO::exec() 返回受修改或删除 SQL 语句影响的行数。如果没有受影响的行，则 PDO::exec() 返回 0。

下面例子依赖 PDO::exec() 的返回值是不正确的，其中受影响行数为 0 的语句会导致调用 die()：

```
<?php
$db->exec() or die(print_r($db->errorInfo(), true));
?>
```

实例

执行一条 DELETE 语句

计算由一条不带 WHERE 字句的 DELETE 语句删除的行数。

```
<?php
$dbh = new PDO('odbc:sample', 'db2inst1', 'ibmdb2');

/*  删除 FRUIT 数据表中满足条件的所有行 */
$count = $dbh->exec("DELETE FROM fruit WHERE colour = 'red'");

/* 返回被删除的行数 */
print("Deleted $count rows.\n");
?>
```

以上例程会输出：

```
Deleted 1 rows.
```

PDO::getAttribute

PDO::getAttribute — 取回一个数据库连接的属性(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
mixed PDO::getAttribute ( int $attribute )
```

此函数（方法）返回一个数据库连接的属性值。 取回 PDOStatement 属性，请参阅 PDOStatement::getAttribute()。

注意有些数据库/驱动可能不支持所有的数据库连接属性。

参数说明：

attribute： PDO::ATTR_* 常量中的一个。下列为应用到数据库连接中的常量：

- *PDO::ATTR_AUTOCOMMIT*
- *PDO::ATTR_CASE*
- *PDO::ATTR_CLIENT_VERSION*
- *PDO::ATTR_CONNECTION_STATUS*
- *PDO::ATTR_DRIVER_NAME*
- *PDO::ATTR_ERRMODE*
- *PDO::ATTR_ORACLE_NULLS*
- *PDO::ATTR_PERSISTENT*
- *PDO::ATTR_PREFETCH*
- *PDO::ATTR_SERVER_INFO*
- *PDO::ATTR_SERVER_VERSION*
- *PDO::ATTR_TIMEOUT*

返回值

成功调用则返回请求的 PDO 属性值。不成功则返回 null。

实例

取回数据库连接属性

```
<?php
$conn = new PDO('odbc:sample', 'db2inst1', 'ibmdb2');
$attributes = array(
    "AUTOCOMMIT", "ERRMODE", "CASE", "CLIENT_VERSION", "CONNECTION_STATUS",
    "ORACLE_NULLS", "PERSISTENT", "PREFETCH", "SERVER_INFO", "SERVER_VERSION",
    "TIMEOUT"
);

foreach ($attributes as $val) {
    echo "PDO::ATTR_$val: ";
    echo $conn->getAttribute(constant("PDO::ATTR_$val")) . "\n";
}
?>
```

PDO::getAvailableDrivers

PDO::getAvailableDrivers — 返回一个可用驱动的数组(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
static array PDO::getAvailableDrivers ( void )
```

```
array pdo_drivers ( void )
```

此函数（方法）返回所有当前可用在 PDO::__construct() 的参数 DSN 中的 PDO 驱动。

返回值

PDO::getAvailableDrivers() 返回一个 包含可用 PDO 驱动名字的数组。如果没有可用的驱动，则返回一个空数组。

实例

一个 PDO::getAvailableDrivers() 的例子

```
<?php
print_r(PDO::getAvailableDrivers());
?>
```

以上例程的输出类似于：

```
Array
(
    [0] => mysql
    [1] => sqlite
)
```

PDO::inTransaction

PDO::inTransaction — 检查是否在一个事务内(PHP 5 >= 5.3.3, Bundled pdo_pgsql)

说明

语法

```
bool PDO::inTransaction ( void )
```

检查驱动内的一个事务当前是否处于激活。此方法仅对支持事务的数据库驱动起作用。

参数

此函数没有参数。

返回值

如果当前事务处于激活，则返回 TRUE ， 否则返回 FALSE 。

PDO::lastInsertId

PDO::lastInsertId — 返回最后插入行的ID或序列值(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
string PDO::lastInsertId ([ string $name = NULL ] )
```

返回最后插入行的ID，或者是一个序列对象最后的值，取决于底层的驱动。比如，PDO_PGSQL() 要求为 name 参数指定序列对象的名称。

注意： 在不同的 PDO 驱动之间，此方法可能不会返回一个有意义或一致的结果，因为底层数据库可能不支持自增字段或序列的概念。

参数

name 应该返回ID的那个序列对象的名称。

返回值

如果没有为参数 name 指定序列名称，PDO::lastInsertId() 则返回一个表示最后插入数据库那一行的行ID的字符串。

如果为参数 name 指定了序列名称，PDO::lastInsertId() 则返回一个表示从指定序列对象取回最后的值的字符串。

如果当前 PDO 驱动不支持此功能，则 PDO::lastInsertId() 触发一个 IM001 SQLSTATE 。

PDO::prepare

PDO::prepare — 准备要执行的SQL语句并返回一个 PDOStatement 对象(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
public PDOStatement PDO::prepare ( string $statement [, array $driver_options = array() ]
```

为 PDOStatement::execute() 方法准备要执行的SQL语句，SQL语句可以包含零个或多个命名（:name）或问号（?）参数标记，参数在SQL执行时会被替换。

你不能在 SQL 语句中同时包含命名（:name）或问号（?）参数标记，只能选择其中一种风格。

预处理 SQL 语句中的参数在使用PDOStatement::execute()方法时会传递真实的参数。

参数

statement 合法的SQL语句。

driver_options 此数组包含一个或多个 key=>value 对来设置 PDOStatement 对象的属性，最常使用到是将PDO::ATTR_CURSOR值设置为PDO::CURSOR_SCROLL来请求一个可滚动游标。

返回值

如果成功，PDO::prepare()返回PDOStatement对象，如果失败返回 FALSE 或抛出异常 PDOException 。

实例

使用命名（:name）参数来准备SQL语句

```
<?php
/* 通过数组值向预处理语句传递值 */
$sql = 'SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour';
$sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY));
$sth->execute(array(':calories' => 150, ':colour' => 'red'));
$red = $sth->fetchAll();
$sth->execute(array(':calories' => 175, ':colour' => 'yellow'));
$yellow = $sth->fetchAll();
?>
```

使用问号（?）参数来准备SQL语句

```
<?php
/* 通过数组值向预处理语句传递值 */
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->execute(array(150, 'red'));
$red = $sth->fetchAll();
$sth->execute(array(175, 'yellow'));
$yellow = $sth->fetchAll();
?>
```

PDO::query

PDO::query — 执行 SQL 语句，返回PDOStatement对象,可以理解为结果集(PHP 5 >= 5.1.0, PECL pdo >= 0.2.0)

说明

语法

```
public PDOStatement PDO::query ( string $statement )
```

```
public PDOStatement PDO::query ( string $statement , int $PDO::FETCH_COLUMN , int $colno
```

```
public PDOStatement PDO::query ( string $statement , int $PDO::FETCH_CLASS , string $clas
```

```
public PDOStatement PDO::query ( string $statement , int $PDO::FETCH_INTO , object $objec
```

PDO::query() 在一个单独的函数中调用并执行 SQL 语句, 返回结果集 (如果有),语句作为一个PDOStatement对象返回。

参数

statement 要执行的SQL语句。

返回值

如果成功，PDO::query()返回PDOStatement对象，如果失败返回 FALSE 。

实例

PDO::query实例

遍历输出结果集：

```
<?php
function getFruit($conn) {
    $sql = 'SELECT name, color, calories FROM fruit ORDER BY name';
    foreach ($conn->query($sql) as $row) {
        print $row['name'] . "\t";
        print $row['color'] . "\t";
        print $row['calories'] . "\n";
    }
}
?>
```

以上输出结果为：

```
apple   red     150
banana  yellow  250
kiwi    brown   75
lemon   yellow  25
orange  orange  300
pear    green   150
watermelon      pink    90
```

PDO::quote

PDO::quote — 为SQL语句中的字符串添加引号。(PHP 5 >= 5.1.0, PECL pdo >= 0.2.1)

说明

语法

```
public string PDO::quote ( string $string [, int $parameter_type = PDO::PARAM_STR ] )
```

PDO::quote() 为SQL语句中的字符串添加引号或者转义特殊字符串。

参数

string 要添加引号的字符串。

parameter_type 为驱动程序提供数据类型。

返回值

返回一个带引号的字符串，理论上可以安全的传递到SQL语句中并执行。如果该驱动程序不支持则返回FALSE。

实例

为普通字符串添加引号

```
<?php
$conn = new PDO('sqlite:/home/lynn/music.sql3');

/* Simple string */
$string = 'Nice';
print "Unquoted string: $string\n";
print "Quoted string: " . $conn->quote($string) . "\n";
?>
```

以上输出结果为：

```
Unquoted string: Nice
Quoted string: 'Nice'
```

转义特殊字符串

```
<?php
$conn = new PDO('sqlite:/home/lynn/music.sql3');

/* Dangerous string */
$string = 'Naughty \' string';
print "Unquoted string: $string\n";
print "Quoted string:" . $conn->quote($string) . "\n";
?>
```

以上例程会输出：

```
Unquoted string: Naughty ' string
Quoted string: 'Naughty '' string'
```

PDO::rollBack

PDO::rollBack — 回滚一个事务(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
bool PDO::rollBack ( void )
```

回滚由 PDO::beginTransaction() 发起的当前事务。如果没有事务激活，将抛出一个 PDOException 异常。

如果数据库被设置成自动提交模式，此函数（方法）在回滚事务之后将恢复自动提交模式。

包括 MySQL 在内的一些数据库， 当在一个事务内有类似删除或创建数据表等 DLL 语句时，会自动导致一个隐式地提交。隐式地提交将无法回滚此事务范围内的任何更改。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

回滚一个事务

下面例子在回滚更改之前开始一个事务并发出两条修改数据库的语句。但在 MySQL 中，DROP TABLE 语句自动提交事务，因此在此事务内的任何更改都不会被回滚。

```
<?php
/* 开始一个事务，关闭自动提交 */
$dbh->beginTransaction();

/* 更改数据库架构和数据  */
$sth = $dbh->exec("DROP TABLE fruit");
$sth = $dbh->exec("UPDATE dessert
    SET name = 'hamburger'");

/*  识别错误且回滚更改  */
$dbh->rollBack();

/*  此时数据库连接恢复到自动提交模式  */
?>
```

PDO::setAttribute

PDO::setAttribute — 设置属性(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
bool PDO::setAttribute ( int $attribute , mixed $value )
```

设置数据库句柄属性。下面列出了一些可用的通用属性；有些驱动可能使用另外的特定属性。

- *PDO::ATTR_CASE*：强制列名为指定的大小写。
  - *PDO::CASE_LOWER*：强制列名小写。
  - *PDO::CASE_NATURAL*：保留数据库驱动返回的列名。
  - *PDO::CASE_UPPER*：强制列名大写。
- *PDO::ATTR_ERRMODE*：错误报告。
  - *PDO::ERRMODE_SILENT*： 仅设置错误代码。
  - *PDO::ERRMODE_WARNING*: 引发 *E_WARNING* 错误
  - *PDO::ERRMODE_EXCEPTION*: 抛出 *exceptions* 异常。
- *PDO::ATTR_ORACLE_NULLS* （在所有驱动中都可用，不仅限于Oracle）： 转换 NULL 和空字符串。
  - *PDO::NULL_NATURAL*: 不转换。
  - *PDO::NULL_EMPTY_STRING*： 将空字符串转换成 `NULL` 。
  - *PDO::NULL_TO_STRING*: 将 NULL 转换成空字符串。
- *PDO::ATTR_STRINGIFY_FETCHES*: 提取的时候将数值转换为字符串。 需要 *bool*。
- *PDO::ATTR_STATEMENT_CLASS*： 设置从PDOStatement派生的用户提供的语句类。 不能用于持久的PDO实例。 需要 *array(string 类名, array(mixed 构造函数的参数))*。

- *PDO::ATTR_TIMEOUT*： 指定超时的秒数。并非所有驱动都支持此选项，这意味着驱动和驱动之间可能会有差异。比如，SQLite等待的时间达到此值后就放弃获取可写锁，但其他驱动可能会将此值解释为一个连接或读取超时的间隔。 需要 int 类型。
- *PDO::ATTR_AUTOCOMMIT* （在OCI，Firebird 以及 MySQL中可用）： 是否自动提交每个单独的语句。
- *PDO::ATTR_EMULATE_PREPARES* 启用或禁用预处理语句的模拟。 有些驱动不支持或有限度地支持本地预处理。使用此设置强制PDO总是模拟预处理语句（如果为 `TRUE` ），或试着使用本地预处理语句（如果为 `FALSE` ）。如果驱动不能成功预处理当前查询，它将总是回到模拟预处理语句上。 需要 *bool* 类型。
- *PDO::MYSQL_ATTR_USE_BUFFERED_QUERY* （在MySQL中可用）： 使用缓冲查询。
- *PDO::ATTR_DEFAULT_FETCH_MODE*： 设置默认的提取模式。关于模式的说明可以在 *PDOStatement::fetch()* 文档找到。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

PDOStatement::bindColumn

PDOStatement::bindColumn — 绑定一列到一个 PHP 变量(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
bool PDOStatement::bindColumn ( mixed $column , mixed &$param [, int $type [, int $maxlen
```

安排一个特定的变量绑定到一个查询结果集中给定的列。每次调用 PDOStatement::fetch() 或 PDOStatement::fetchAll() 都将更新所有绑定到列的变量。

注意：在语句执行前 PDO 有关列的信息并非总是可用，可移植的应用应在 PDOStatement::execute() 之后 调用此函数（方法）。 但是，当使用 PgSQL 驱动 时，要想能绑定一个 LOB 列作为流，应用程序必须在调用 PDOStatement::execute() 之前 调用此方法，否则大对象 OID 作为一个整数返回。

参数

column 结果集中的列号（从1开始索引）或列名。如果使用列名，注意名称应该与由驱动返回的列名大小写保持一致。

param 将绑定到列的 PHP 变量名称

type 通过 PDO::PARAM_* 常量指定的参数的数据类型。

maxlen 预分配提示。

driverdata 驱动的可选参数。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

把结果集输出绑定到 PHP 变量

绑定结果集中的列到PHP变量是一种使每行包含的数据在应用程序中立即可用的有效方法。下面的例子演示了 PDO 怎样用多种选项和缺省值绑定和检索列。

```
<?php
function readData($dbh) {
  $sql = 'SELECT name, colour, calories FROM fruit';
  try {
    $stmt = $dbh->prepare($sql);
    $stmt->execute();

    /*  通过列号绑定  */
    $stmt->bindColumn(1, $name);
    $stmt->bindColumn(2, $colour);

    /*  通过列名绑定  */
    $stmt->bindColumn('calories', $cals);

    while ($row = $stmt->fetch(PDO::FETCH_BOUND)) {
      $data = $name . "\t" . $colour . "\t" . $cals . "\n";
      print $data;
    }
  }
  catch (PDOException $e) {
    print $e->getMessage();
  }
}
readData($dbh);
?>
```

以上例程会输出：

```
apple   red     150
banana  yellow  175
kiwi    green   75
orange  orange  150
mango   red     200
strawberry      red     25
```

PDOStatement::bindParam

PDOStatement::bindParam — 绑定一个参数到指定的变量名(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
bool PDOStatement::bindParam ( mixed $parameter , mixed &$variable [, int $data_type = PD
```

绑定一个PHP变量到用作预处理的SQL语句中的对应命名占位符或问号占位符。 不同于 PDOStatement::bindValue()，此变量作为引用被绑定，并只在 PDOStatement::execute() 被调用的时候才取其值。

大多数参数是输入参数，即，参数以只读的方式用来建立查询。一些驱动支持调用存储过程并作为输出参数返回数据，一些支持作为输入/输出参数，既发送数据又接收更新后的数据。

参数

parameter 参数标识符。对于使用命名占位符的预处理语句，应是类似 :name 形式的参数名。对于使用问号占位符的预处理语句，应是以1开始索引的参数位置。

variable 绑定到 SQL 语句参数的 PHP 变量名。

data_type 使用 PDO::PARAM_* 常量明确地指定参数的类型。要从一个存储过程中返回一个 INOUT 参数，需要为 data_type 参数使用按位或操作符去设置 PDO::PARAM_INPUT_OUTPUT 位。

length 预分配提示。

driverdata 数据类型的长度。为表明参数是一个存储过程的 OUT 参数，必须明确地设置此长度。

driver_options

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

执行一条使用命名占位符的预处理语句

```
<?php
/* 通过绑定的 PHP 变量执行一条预处理语句  */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindParam(':colour', $colour, PDO::PARAM_STR, 12);
$sth->execute();
?>
```

执行一条使用问号占位符的预处理语句

```
<?php
/*  通过绑定的 PHP 变量执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->bindParam(2, $colour, PDO::PARAM_STR, 12);
$sth->execute();
?>
```

使用 INOUT 参数调用一个存储过程

```
<?php
/* 使用 INOUT 参数调用一个存储过程 */
$colour = 'red';
$sth = $dbh->prepare('CALL puree_fruit(?)');
$sth->bindParam(1, $colour, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 12);
$sth->execute();
print("After pureeing fruit, the colour is: $colour");
?>
```

PDOStatement::bindValue

PDOStatement::bindValue — 把一个值绑定到一个参数(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
bool PDOStatement::bindValue ( mixed $parameter , mixed $value [, int $data_type = PDO::P
```

绑定一个值到用作预处理的 SQL 语句中的对应命名占位符或问号占位符。

参数

parameter 参数标识符。对于使用命名占位符的预处理语句，应是类似 :name 形式的参数名。对于使用问号占位符的预处理语句，应是以1开始索引的参数位置。

value 绑定到参数的值

data_type 使用 PDO::PARAM_* 常量明确地指定参数的类型。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

执行一条使用命名占位符的预处理语句

```
<?php
/* 通过绑定的 PHP 变量执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$sth->bindValue(':calories', $calories, PDO::PARAM_INT);
$sth->bindValue(':colour', $colour, PDO::PARAM_STR);
$sth->execute();
?>
```

执行一条使用问号占位符的预处理语句

```
<?php
/* 通过绑定的 PHP 变量执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->bindValue(1, $calories, PDO::PARAM_INT);
$sth->bindValue(2, $colour, PDO::PARAM_STR);
$sth->execute();
?>
```

PDOStatement::closeCursor

PDOStatement::closeCursor — 关闭游标，使语句能再次被执行。(PHP 5 >= 5.1.0, PECL pdo >= 0.9.0)

说明

语法

```
bool PDOStatement::closeCursor ( void )
```

PDOStatement::closeCursor() 释放到数据库服务的连接，以便发出其他 SQL 语句，但使语句处于一个可以被再次执行的状态。

当上一个执行的 PDOStatement 对象仍有未取行时，此方法对那些不支持再执行一个 PDOStatement 对象的数据库驱动非常有用。 如果数据库驱动受此限制，则可能出现失序错误的问题。

PDOStatement::closeCursor() 要么是一个可选驱动的特有方法（效率最高）来实现，要么是在没有驱动特定的功能时作为一般的PDO 备用来实现。一般的备用语义上与下面的 PHP 代码相同：

```
<?php
do {
    while ($stmt->fetch())
        ;
    if (!$stmt->nextRowset())
        break;
} while (true);
?>
```

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

一个 **PDOStatement::closeCursor()** 的例子

在下面例子中，$stmt PDOStatement 对象返回多行，但应用程序只取第一行，让 PDOStatement 对象处于一个有未取行的状态。为确保应用程序对所有数据库驱动都能正常运行，在执行 $otherStmt PDOStatement 对象前，$stmt 调用一次 PDOStatement::closeCursor() 。

```
<?php
/* 创建一个 PDOStatement 对象 */
$stmt = $dbh->prepare('SELECT foo FROM bar');

/* 创建第二个 PDOStatement 对象 */
$otherStmt = $dbh->prepare('SELECT foobaz FROM foobar');

/* 执行第一条语句 */
$stmt->execute();

/*  从结果集中只取出第一行 */
$stmt->fetch();

/* The following call to closeCursor() may be required by some drivers */
$stmt->closeCursor();

/*  现在可以执行第二条语句了 */
$otherStmt->execute();
?>
```

PDOStatement::columnCount

PDOStatement::columnCount — 返回结果集中的列数。(PHP 5 >= 5.1.0, PECL pdo >= 0.2.0)

说明

语法

```
int PDOStatement::columnCount ( void )
```

使用 PDOStatement::columnCount() 返回由 PDOStatement 对象代表的结果集中的列数。

如果是由 PDO::query() 返回的 PDOStatement 对象，则列数计算立即可用。

如果是由 PDO::prepare() 返回的 PDOStatement 对象，则在调用 PDOStatement::execute() 之前都不能准确地计算出列数。

返回值

返回由 PDOStatement 对象代表的结果集中的列数。如果没有结果集，则 PDOStatement::columnCount() 返回 0。

实例

计算列数

下面例子演示如何使用 PDOStatement::columnCount() 操作一个结果集和一个空集。

```
<?php
$dbh = new PDO('odbc:sample', 'db2inst1', 'ibmdb2');

$sth = $dbh->prepare("SELECT name, colour FROM fruit");

/*  计算一个（不存在）的结果集中的列数 */
$colcount = $sth->columnCount();
print("Before execute(), result set has $colcount columns (should be 0)\n");

$sth->execute();

/* 计算结果集中的列数 */
$colcount = $sth->columnCount();
print("After execute(), result set has $colcount columns (should be 2)\n");

?>
```

以上例程会输出：

```
Before execute(), result set has 0 columns (should be 0)
After execute(), result set has 2 columns (should be 2)
```

PDOStatement::debugDumpParams

PDOStatement::debugDumpParams — 打印一条 SQL 预处理命令(PHP 5 >= 5.1.0, PECL pdo >= 0.9.0)

说明

语法

```
bool PDOStatement::debugDumpParams ( void )
```

直接打印出一条预处理语句包含的信息。提供正在使用的 SQL 查询、所用参数（Params）的数目、参数的清单、参数名、用一个整数表示的参数类型（paramtype）、键名或位置、值、以及在查询中的位置（如果当前 POD 驱动不支持，则为-1）。

此为一个用于调试的功能，在正常输出的情况下直接输出数据。

提示： 和直接将结果输出到浏览器一样，可使用输出控制函数来捕获当前函数的输出，然后(例如)保存到一个 string 中。

只打印此时此刻语句中的参数。额外的参数不存储在语句中，也就不会被输出。

返回值

没有返回值。

实例

PDOStatement::debugDumpParams() 使用命名参数的例子

```
<?php
/* 通过绑定 PHP 变量执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindValue(':colour', $colour, PDO::PARAM_STR, 12);
$sth->execute();

$sth->debugDumpParams();

?>
```

以上例程会输出：

```
SQL: [96] SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour
Params:  2
Key: Name: [9] :calories
paramno=-1
name=[9] ":calories"
is_param=1
param_type=1
Key: Name: [7] :colour
paramno=-1
name=[7] ":colour"
is_param=1
param_type=2
```

PDOStatement::debugDumpParams() 使用未命名参数的例子

```
<?php

/* 通过绑定 PHP 变量执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$name = 'apple';

$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->bindValue(2, $colour, PDO::PARAM_STR);
$sth->execute();

$sth->debugDumpParams();

?>
```

以上例程会输出：

```
SQL: [82] SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?
Params:  2
Key: Position #0:
paramno=0
name=[0] ""
is_param=1
param_type=1
Key: Position #1:
paramno=1
name=[0] ""
is_param=1
param_type=2
```

PDOStatement::errorCode

PDOStatement::errorCode — 获取跟上一次语句句柄操作相关的 SQLSTATE(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
string PDOStatement::errorCode ( void )
```

与 PDO::errorCode() 相同，只是 PDOStatement::errorCode() 只取回 PDOStatement 对象执行操作中的错误码。

返回值

没有返回值。

实例

取回一个 SQLSTATE 码

```
<?php
/* 引发一个错误 --  BONES 数据表不存在 */
$err = $dbh->prepare('SELECT skull FROM bones');
$err->execute();

echo "\nPDOStatement::errorCode(): ";
print $err->errorCode();
?>
```

以上例程会输出：

```
PDOStatement::errorCode(): 42S02
```

PDOStatement::errorInfo

PDOStatement::errorInfo — 获取跟上一次语句句柄操作相关的扩展错误信息(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
array PDOStatement::errorInfo ( void )
```

PDOStatement::errorInfo() 返回一个关于上一次语句句柄执行操作的错误信息的数组 。该数组包含下列字段：

元素	信息
0	SQLSTATE 错误码（一个由5个字母或数字组成的在 ANSI SQL 标准中定义的标识符）。
1	具体驱动错误码。
2	具体驱动错误信息。

实例

显示连接到DB2数据库的 **PDO_ODBC** 连接的 **errorInfo()** 的字段

```
<?php
/* 激发一个错误 --  BONES 数据表不存在 */
$sth = $dbh->prepare('SELECT skull FROM bones');
$sth->execute();

echo "\nPDOStatement::errorInfo():\n";
$arr = $sth->errorInfo();
print_r($arr);
?>
<pre>
PDOStatement::errorCode(): 42S02
```

以上例程会输出：

```
PDOStatement::errorInfo():
Array
(
    [0] => 42S02
    [1] => -204
    [2] => [IBM][CLI Driver][DB2/LINUX] SQL0204N  "DANIELS.BONES" is an undefined name.
)
```




PDOStatement::execute

PDOStatement::execute — 执行一条预处理语句(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
bool PDOStatement::execute ([ array $input_parameters ] )
```

执行预处理过的语句。如果预处理过的语句含有参数标记，必须选择下面其中一种做法：

- 调用 PDOStatement::bindParam() 绑定 PHP 变量到参数标记：如果有的话，通过关联参数标记绑定的变量来传递输入值和取得输出值
- 或传递一个只作为输入参数值的数组

参数

input_parameters

一个元素个数和将被执行的 SQL 语句中绑定的参数一样多的数组。所有的值作为 PDO::PARAM_STR 对待。

不能绑定多个值到一个单独的参数；比如，不能绑定两个值到 IN（）子句中一个单独的命名参数。

绑定的值不能超过指定的个数。如果在 input_parameters 中存在比 PDO::prepare() 预处理的 SQL 指定的多的键名，则此语句将会失败并发出一个错误。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

执行一条绑定变量的预处理语句

```
<?php
/*  通过绑定 PHP 变量执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindParam(':colour', $colour, PDO::PARAM_STR, 12);
$sth->execute();
?>
```

使用一个含有插入值的数组执行一条预处理语句（命名参数）

```
<?php
/* 通过传递一个含有插入值的数组执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$sth->execute(array(':calories' => $calories, ':colour' => $colour));
?>
```

使用一个含有插入值的数组执行一条预处理语句（占位符）

```
<?php
/*  通过传递一个插入值的数组执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->execute(array($calories, $colour));
?>
```

执行一条问号占位符的预处理语句

```
<?php
/* 通过绑定 PHP 变量执行一条预处理语句 */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->bindParam(2, $colour, PDO::PARAM_STR, 12);
$sth->execute();
?>
```

使用数组执行一条含有 IN 子句的预处理语句

```
<?php
/*  使用一个数组的值执行一条含有 IN 子句的预处理语句 */
$params = array(1, 21, 63, 171);
/*  创建一个填充了和params相同数量占位符的字符串 */
$place_holders = implode(',', array_fill(0, count($params), '?'));

/*
    对于 $params 数组中的每个值，要预处理的语句包含足够的未命名占位符 。
    语句被执行时， $params 数组中的值被绑定到预处理语句中的占位符。
    这和使用 PDOStatement::bindParam() 不一样，因为它需要一个引用变量。
    PDOStatement::execute() 仅作为通过值绑定的替代。
*/
$sth = $dbh->prepare("SELECT id, name FROM contacts WHERE id IN ($place_holders)");
$sth->execute($params);
?>
```

PDOStatement::fetch

PDOStatement::fetch — 从结果集中获取下一行(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
mixed PDOStatement::fetch ([ int $fetch_style [, int $cursor_orientation = PDO::FETCH_ORI
```

从一个 PDOStatement 对象相关的结果集中获取下一行。fetch_style 参数决定 POD 如何返回行。

参数

fetch_style

控制下一行如何返回给调用者。此值必须是 PDO::FETCH_* 系列常量中的一个，缺省为 PDO::ATTR_DEFAULT_FETCH_MODE 的值 （默认为 PDO::FETCH_BOTH ）。

- *PDO::FETCH_ASSOC*：返回一个索引为结果集列名的数组
- *PDO::FETCH_BOTH*（默认）：返回一个索引为结果集列名和以0开始的列号的数组
- *PDO::FETCH_BOUND*：返回 `TRUE` ，并分配结果集中的列值给 PDOStatement::bindColumn() 方法绑定的 PHP 变量。
- *PDO::FETCH_CLASS*：返回一个请求类的新实例，映射结果集中的列名到类中对应的属性名。如果 `fetch_style` 包含 PDO::FETCH*CLASSTYPE*（例如：_*PDO::FETCH_CLASS* | *PDO::FETCH_CLASSTYPE*），则类名由第一列的值决定
- *PDO::FETCH_INTO*：更新一个被请求类已存在的实例，映射结果集中的列到类中命名的属性
- *PDO::FETCH_LAZY*：结合使用 *PDO::FETCH_BOTH* 和 *PDO::FETCH_OBJ*，创建供用来访问的对象变量名
- *PDO::FETCH_NUM*：返回一个索引为以0开始的结果集列号的数组
- *PDO::FETCH_OBJ*：返回一个属性名对应结果集列名的匿名对象

cursor_orientation 对于 一个 PDOStatement 对象表示的可滚动游标，该值决定了哪一行将被返回给调用者。此值必须是 PDO::FETCH*ORI** 系列常量中的一个，默认为 PDO::FETCH_ORI_NEXT。要想让 PDOStatement 对象使用可滚动游标，必须在用 PDO::prepare() 预处理SQL语句时，设置 PDO::ATTR_CURSOR 属性为 PDO::CURSOR_SCROLL。

offset 对于一个 cursor_orientation 参数设置为 PDO::FETCH_ORI_ABS 的PDOStatement 对象代表的可滚动游标，此值指定结果集中想要获取行的绝对行号。 对于一个 cursor_orientation 参数设置为 PDO::FETCH_ORI_REL 的PDOStatement 对象代表的可滚动游标，此值指定想要获取行相对于调用 PDOStatement::fetch() 前游标的位置

返回值

此函数（方法）成功时返回的值依赖于提取类型。在所有情况下，失败都返回 FALSE 。

实例

用不同的提取方式获取行

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();

/* 运用 PDOStatement::fetch 风格 */
print("PDO::FETCH_ASSOC: ");
print("Return next row as an array indexed by column name\n");
$result = $sth->fetch(PDO::FETCH_ASSOC);
print_r($result);
print("\n");

print("PDO::FETCH_BOTH: ");
print("Return next row as an array indexed by both column name and number\n");
$result = $sth->fetch(PDO::FETCH_BOTH);
print_r($result);
print("\n");

print("PDO::FETCH_LAZY: ");
print("Return next row as an anonymous object with column names as properties\n");
$result = $sth->fetch(PDO::FETCH_LAZY);
print_r($result);
print("\n");

print("PDO::FETCH_OBJ: ");
print("Return next row as an anonymous object with column names as properties\n");
$result = $sth->fetch(PDO::FETCH_OBJ);
print $result->NAME;
print("\n");
?>
```

以上实例会输出：

```
PDO::FETCH_ASSOC: Return next row as an array indexed by column name
Array
(
    [NAME] => apple
    [COLOUR] => red
)

PDO::FETCH_BOTH: Return next row as an array indexed by both column name and number
Array
(
    [NAME] => banana
    [0] => banana
    [COLOUR] => yellow
    [1] => yellow
)

PDO::FETCH_LAZY: Return next row as an anonymous object with column names as properties
PDORow Object
(
    [NAME] => orange
    [COLOUR] => orange
)

PDO::FETCH_OBJ: Return next row as an anonymous object with column names as properties
kiwi
```

使用一个可滚动游标获取行

```
<?php
function readDataForwards($dbh) {
  $sql = 'SELECT hand, won, bet FROM mynumbers ORDER BY BET';
  try {
    $stmt = $dbh->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
    $stmt->execute();
    while ($row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_NEXT)) {
      $data = $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";
      print $data;
    }
    $stmt = null;
  }
  catch (PDOException $e) {
    print $e->getMessage();
  }
}
function readDataBackwards($dbh) {
  $sql = 'SELECT hand, won, bet FROM mynumbers ORDER BY bet';
  try {
    $stmt = $dbh->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_LAST);
    do {
      $data = $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";
      print $data;
    } while ($row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_PRIOR));
    $stmt = null;
  }
  catch (PDOException $e) {
    print $e->getMessage();
  }
}

print "Reading forwards:\n";
readDataForwards($conn);

print "Reading backwards:\n";
readDataBackwards($conn);
?>
```

以上实例会输出：

```
Reading forwards:
21    10    5
16    0     5
19    20    10

Reading backwards:
19    20    10
16    0     5
21    10    5
```

PDOStatement::fetchAll

PDOStatement::fetchAll — 返回一个包含结果集中所有行的数组(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
array PDOStatement::fetchAll ([ int $fetch_style [, mixed $fetch_argument [, array $ctor_
```

参数

fetch_style

控制下一行如何返回给调用者。此值必须是 PDO::FETCH_* 系列常量中的一个，缺省为 PDO::ATTR_DEFAULT_FETCH_MODE 的值 （默认为 PDO::FETCH_BOTH ）。

想要返回一个包含结果集中单独一列所有值的数组，需要指定 PDO::FETCH_COLUMN 。通过指定 column-index 参数获取想要的列。

想要获取结果集中单独一列的唯一值，需要将 PDO::FETCH_COLUMN 和 PDO::FETCH_UNIQUE 按位或。

想要返回一个根据指定列把值分组后的关联数组，需要将 PDO::FETCH_COLUMN 和 PDO::FETCH_GROUP 按位或。

fetch_argument 根据 fetch_style 参数的值，此参数有不同的意义：

- `PDO::FETCH_COLUMN` ：返回指定以0开始索引的列。
- `PDO::FETCH_CLASS` ：返回指定类的实例，映射每行的列到类中对应的属性名。
- `PDO::FETCH_FUNC` ：将每行的列作为参数传递给指定的函数，并返回调用函数后的结果。

ctor_args 当 fetch_style 参数为 PDO::FETCH_CLASS 时，自定义类的构造函数的参数。

返回值

PDOStatement::fetchAll() 返回一个包含结果集中所有剩余行的数组。此数组的每一行要么是一个列值的数组，要么是属性对应每个列名的一个对象。

使用此方法获取大结果集将导致系统负担加重且可能占用大量网络资源。与其取回所有数据后用PHP来操作，倒不如考虑使用数据库服务来处理结果集。例如，在取回数据并通过PHP处理前，在 SQL 中使用 WHERE 和 ORDER BY 子句来限定结果。

实例

获取结果集中所有剩余的行

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();

/* 获取结果集中所有剩余的行 */
print("Fetch all of the remaining rows in the result set:\n");
$result = $sth->fetchAll();
print_r($result);
?>
```

以上实例的输出为：

```
Fetch all of the remaining rows in the result set:
Array
(
    [0] => Array
        (
            [NAME] => pear
            [0] => pear
            [COLOUR] => green
            [1] => green
        )

    [1] => Array
        (
            [NAME] => watermelon
            [0] => watermelon
            [COLOUR] => pink
            [1] => pink
        )

)
```

获取结果集中单独一列的所有值

下面例子演示了如何从一个结果集中返回单独一列所有的值，尽管 SQL 语句自身可能返回每行多列。

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();

/* 获取第一列所有值 */
$result = $sth->fetchAll(PDO::FETCH_COLUMN, 0);
var_dump($result);
?>
```

以上实例的输出为：

```
Array(3)
(
    [0] =>
    string(5) => apple
    [1] =>
    string(4) => pear
    [2] =>
    string(10) => watermelon
)
```

根据单独的一列把所有值分组

下面例子演示了如何返回一个根据结果集中指定列的值分组的关联数组。该数组包含三个键：返回的 apple 和 pear 数组包含了两种不同的颜色，而返回的 watermelon 数组仅包含一种颜色。

```
<?php
$insert = $dbh->prepare("INSERT INTO fruit(name, colour) VALUES (?, ?)");
$insert->execute(array('apple', 'green'));
$insert->execute(array('pear', 'yellow'));

$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();

/* 根据第一列分组  */
var_dump($sth->fetchAll(PDO::FETCH_COLUMN|PDO::FETCH_GROUP));
?>
```

以上实例的输出为：

```
array(3) {
  ["apple"]=>
  array(2) {
    [0]=>
    string(5) "green"
    [1]=>
    string(3) "red"
  }
  ["pear"]=>
  array(2) {
    [0]=>
    string(5) "green"
    [1]=>
    string(6) "yellow"
  }
  ["watermelon"]=>
  array(1) {
    [0]=>
    string(5) "green"
  }
}
```

每行结果实例化一个类

下面列子演示了 PDO::FETCH_CLASS 获取风格的行为。

```
<?php
class fruit {
    public $name;
    public $colour;
}

$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();

$result = $sth->fetchAll(PDO::FETCH_CLASS, "fruit");
var_dump($result);
?>
```

以上实例的输出为：

```
array(3) {
  [0]=>
  object(fruit)#1 (2) {
    ["name"]=>
    string(5) "apple"
    ["colour"]=>
    string(5) "green"
  }
  [1]=>
  object(fruit)#2 (2) {
    ["name"]=>
    string(4) "pear"
    ["colour"]=>
    string(6) "yellow"
  }
  [2]=>
  object(fruit)#3 (2) {
    ["name"]=>
    string(10) "watermelon"
    ["colour"]=>
    string(4) "pink"
  }
}
```

每行调用一次函数

下面列子演示了 PDO::FETCH_FUNC 获取风格的行为。

```
<?php
function fruit($name, $colour) {
    return "{$name}: {$colour}";
}

$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();

$result = $sth->fetchAll(PDO::FETCH_FUNC, "fruit");
var_dump($result);
?>
```

以上实例的输出为：

```
array(3) {
  [0]=>
  string(12) "apple: green"
  [1]=>
  string(12) "pear: yellow"
  [2]=>
  string(16) "watermelon: pink"
}
```

PDOStatement::fetchColumn

PDOStatement::fetchColumn — 从结果集中的下一行返回单独的一列。(PHP 5 >= 5.1.0, PECL pdo >= 0.9.0)

说明

语法

```
string PDOStatement::fetchColumn ([ int $column_number = 0 ] )
```

从结果集中的下一行返回单独的一列，如果没有了，则返回 FALSE 。

参数

column_number

你想从行里取回的列的索引数字（以0开始的索引）。如果没有提供值，则 PDOStatement::fetchColumn() 获取第一列。

返回值

PDOStatement::fetchColumn() 从结果集中的下一行返回单独的一列。

注意：如果使用 PDOStatement::fetchColumn() 取回数据，则没有办法返回同一行的另外一列。**

实例

返回下一行的第一列

```
<?php
$sth = $dbh->prepare("SELECT name, colour FROM fruit");
$sth->execute();

/* 从结果集中的下一行获取第一列  */
print("从结果集中的下一行获取第一列：\n");
$result = $sth->fetchColumn();
print("name = $result\n");

print("从结果集中的下一行获取第二列：\n");
$result = $sth->fetchColumn(1);
print("colour = $result\n");
?>
```

以上实例会输出：

```
从结果集中的下一行获取第一列：
name = lemon
从结果集中的下一行获取第二列：
colour = red
```

PDOStatement::fetchObject

PDOStatement::fetchObject — 获取下一行并作为一个对象返回。(PHP 5 >= 5.1.0, PECL pdo >= 0.2.4)

说明

语法

```
mixed PDOStatement::fetchObject ([ string $class_name = "stdClass" [, array $ctor_args ]]
```

获取下一行并作为一个对象返回。此函数（方法）是使用 PDO::FETCH_CLASS 或 PDO::FETCH_OBJ 风格的 PDOStatement::fetch() 的一种替代。

参数

class_name

创建类的名称。

ctor_args

此数组的元素被传递给构造函数。

返回值

返回一个属性名对应于列名的所要求类的实例， 或者在失败时返回 FALSE。

PDOStatement::getAttribute

PDOStatement::getAttribute — 检索一个语句属性(PHP 5 >= 5.1.0, PECL pdo >= 0.2.0)

说明

语法

```
mixed PDOStatement::getAttribute ( int $attribute )
```

得到语句的一个属性。当前，不存在通用的属性，只有驱动特定的属性：

- PDO::ATTR_CURSOR_NAME （Firebird 和 ODBC 特性）：获取 UPDATE ... WHERE CURRENT OF 的游标名称。

返回值

返回属性值。

PDOStatement::getColumnMeta

PDOStatement::getColumnMeta — 返回结果集中一列的元数据(PHP 5 >= 5.1.0, PECL pdo >= 0.2.0)

说明

语法

```
array PDOStatement::getColumnMeta ( int $column )
```

检索一个在结果集中以0开始索引的列的元数据作为一个关联数组。

注意：此函数是实验性的。此函数的表象，包括名称及其相关文档都可能在未来的 PHP 发布版本中未通知就被修改。使用本函数风险自担 。

注意：并非所有 PDO 驱动都支持 PDOStatement::getColumnMeta()。

参数

column 结果集中以0开始索引的列。

返回值

返回一个关联数组，它包含了下列表示一个单独列的元数据的值：

列的元数据 | 名称 | 值 | | --- | --- | | *native_type* | 用于表示列值的 PHP 原生类型。 | | *driver:decl_type* | 在数据库中用于表示列值的 SQL 类型。如果结果集中的列是一个函数的结果，则该值不能被 **PDOStatement::getColumnMeta()** 返回。 | | *flags* | 任何设置于此列的标记。 | | *name* | 通过数据库返回的列名。 | | *table* | 通过数据库返回的该列的表名 | | *len* | 该列的长度。除浮点小数外通常为 *-1* | | *precision* | 该列的数值精度。除浮点小数外通常为 *0* 。 | | *pdo_type* | 以 *PDO::PARAM**_ 常量为代表的列类型。 |

实例

检索列的元数据

下面例子展示了在一个PDO_SQLITE中，检索一个通过函数（COUNT）生成单独列的元数据的结果。

```
<?php
$select = $DB->query('SELECT COUNT(*) FROM fruit');
$meta = $select->getColumnMeta(0);
var_dump($meta);
?>
```

以上实例输出：

```
array(6) {
  ["native_type"]=>
  string(7) "integer"
  ["flags"]=>
  array(0) {
  }
  ["name"]=>
  string(8) "COUNT(*)"
  ["len"]=>
  int(-1)
  ["precision"]=>
  int(0)
  ["pdo_type"]=>
  int(2)
}
```

PDOStatement::nextRowset

PDOStatement::nextRowset — 在一个多行集语句句柄中推进到下一个行集(PHP 5 >= 5.1.0, PECL pdo >= 0.2.0)

说明

语法

```
bool PDOStatement::nextRowset ( void )
```

一些数据库服务支持返回一个以上行集（也被称为结果集）的存储过程。

PDOStatement::nextRowset() 使你能够结合一个 PDOStatement 对象访问第二个以及后续的行集。上述的每个行集可以有不同的列集合。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

获取由一个存储过程返回的多个行集

下面例子展示了怎样调用一个存储过程，返回三个行集的 MULTIPLE_ROWSETS 。用一个 do / while 循环来循环调用 PDOStatement::nextRowset() 方法， 当不再有行集返回时返回 false 并结束循环。

```
<?php
$sql = 'CALL multiple_rowsets()';
$stmt = $conn->query($sql);
$i = 1;
do {
    $rowset = $stmt->fetchAll(PDO::FETCH_NUM);
    if ($rowset) {
        printResultSet($rowset, $i);
    }
    $i++;
} while ($stmt->nextRowset());

function printResultSet(&$rowset, $i) {
    print "Result set $i:\n";
    foreach ($rowset as $row) {
        foreach ($row as $col) {
            print $col . "\t";
        }
        print "\n";
    }
    print "\n";
}
?>
```

以上实例输出：

```
Result set 1:
apple    red
banana   yellow

Result set 2:
orange   orange    150
banana   yellow    175

Result set 3:
lime     green
apple    red
banana   yellow
```

PDOStatement::rowCount

PDOStatement::rowCount — 返回受上一个 SQL 语句影响的行数(PHP 5 >= 5.1.0, PECL pdo >= 0.1.0)

说明

语法

```
int PDOStatement::rowCount ( void )
```

PDOStatement::rowCount() 返回上一个由对应的 PDOStatement 对象执行DELETE、INSERT、或 UPDATE 语句受影响的行数。

如果上一条由相关 PDOStatement 执行的 SQL 语句是一条 SELECT 语句，有些数据可能返回由此语句返回的行数。但这种方式不能保证对所有数据有效，且对于可移植的应用不应依赖于此方式。

返回值

返回行数。

实例

返回删除的行数

PDOStatement::rowCount() 返回受 DELETE、INSERT、 或 UPDATE 语句影响的行数。

```
<?php
/*  从 FRUIT 数据表中删除所有行 */
$del = $dbh->prepare('DELETE FROM fruit');
$del->execute();

/*  返回被删除的行数 */
print("Return number of rows that were deleted:\n");
$count = $del->rowCount();
print("Deleted $count rows.\n");
?>
```

以上实例输出：

```
Return number of rows that were deleted:
Deleted 9 rows.
```

计算由一个 SELECT 语句返回的行数

对于大多数数据库，PDOStatement::rowCount() 不能返回受一条 SELECT 语句影响的行数。替代的方法是，使用 PDO::query() 来发出一条和原打算中的SELECT语句有相同条件表达式的 SELECT COUNT(*) 语句，然后用 PDOStatement::fetchColumn() 来取得返回的行数。这样应用程序才能正确执行。

```
<?php
$sql = "SELECT COUNT(*) FROM fruit WHERE calories > 100";
if ($res = $conn->query($sql)) {

    /* 检查符合 SELECT 语句的行数 */
  if ($res->fetchColumn() > 0) {

        /* 发出一条真正的 SELECT 语句并操作返回的结果 */
         $sql = "SELECT name FROM fruit WHERE calories > 100";
       foreach ($conn->query($sql) as $row) {
           print "Name: " .  $row['NAME'] . "\n";
         }
    }
    /* 没有匹配的行 -- 执行其他 */
  else {
      print "No rows matched the query.";
    }
}

$res = null;
$conn = null;
?>
```

以上实例输出结果为：

```
apple
banana
orange
pear
```

PDOStatement::setAttribute

PDOStatement::setAttribute — 设置一个语句属性(PHP 5 >= 5.1.0, PECL pdo >= 0.2.0)

说明

语法

```
bool PDOStatement::setAttribute ( int $attribute , mixed $value )
```

给语句设置一个属性。当前，没有通用的属性可以设置，只有驱动特定的属性：

- *PDO::ATTR_CURSOR_NAME* （Firebird 和 ODBC 特性）： 为 *UPDATE ... WHERE CURRENT OF* 设置游标名称。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

PDOStatement::setFetchMode

PDOStatement::setFetchMode — 为语句设置默认的获取模式。(PHP 5 >= 5.1.0, PECL pdo >= 0.2.0)

说明

语法

```
bool PDOStatement::setFetchMode ( int $mode )
```

```
bool PDOStatement::setFetchMode ( int $PDO::FETCH_COLUMN , int $colno )
```

```
bool PDOStatement::setFetchMode ( int $PDO::FETCH_CLASS , string $classname , array $ctor
```

```
bool PDOStatement::setFetchMode ( int $PDO::FETCH_INTO , object $object )
```

参数

mode 获取模式必须是 PDO::FETCH_* 系列常量中的一个。

colno 列号。

classname 类名。

ctorargs 构造函数参数。

object 对象。

返回值

成功时返回 TRUE， 或者在失败时返回 FALSE。

实例

设置获取模式

下面的例子示范如何用 PDOStatement::setFetchMode() 来为一个 PDOStatement 对象更改默认的获取模式。

```
<?php
$sql = 'SELECT name, colour, calories FROM fruit';
try {
  $stmt = $dbh->query($sql);
  $result = $stmt->setFetchMode(PDO::FETCH_NUM);
  while ($row = $stmt->fetch()) {
    print $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";
  }
}
catch (PDOException $e) {
  print $e->getMessage();
}
?>
```

以上实例输出为：

```
apple   red     150
banana  yellow  250
orange  orange  300
kiwi    brown   75
lemon   yellow  25
pear    green   150
watermelon      pink    90
```

PHP SimpleXML 函数

PHP SimpleXML 简介

SimpleXML 函数允许您把 XML 转换为对象。

通过普通的属性选择器或数组迭代器，可以处理这个对象，就像处理任何其他对象一样。

其中的一些函数需要最新的 PHP 版本。

安装

SimpleXML 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP SimpleXML 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
__construct()	创建一个新的 SimpleXMLElement 对象。	5
addAttribute()	给 SimpleXML 元素添加一个属性。	5
addChild()	给 SimpleXML 元素添加一个子元素。	5
asXML()	从 SimpleXML 元素获取 XML 字符串。	5
attributes()	获取 SimpleXML 元素的属性。	5
children()	获取指定节点的子。	5
getDocNamespaces()	获取 XML 文档的命名空间。	5
getName()	获取 SimpleXML 元素的名称。	5
getNamespaces()	从 XML 数据获取命名空间。	5
registerXPathNamespace()	为下一次 XPath 查询创建命名空间语境。	5
simplexml_import_dom()	从 DOM 节点获取 SimpleXMLElement 对象。	5
simplexml_load_file()	从 XML 文档获取 SimpleXMLElement 对象。	5
simplexml_load_string()	从 XML 字符串获取 SimpleXMLElement 对象。	5
xpath()	对 XML 数据运行 XPath 查询。	5

PHP SimpleXML 常量

无。

PHP __construct() 函数

定义和用法

__construct() 函数创建一个新的 SimpleXMLElement 对象。

如果成功，则该函数返回一个对象。如果失败，则返回 false。

语法

```
__construct(data,options,is_url,ns,is_prefix)
```

参数	描述
data	必需。形式良好的 XML 字符串或 XML 文档的路径或 URL。
options	可选。规定附加的 Libxml 参数。
is_url	可选。规定 data 参数是否是 URL。默认是 false。
ns	可选。
is_prefix	可选。

返回值

返回一个表示数据的 SimpleXMLElement 对象。

例子

```
<?php
$xmlstring = <<<XML
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
XML;

$xml = new SimpleXMLElement($xmlstring);

echo $xml->body[0];
?>
```

输出类似：

```
Don't forget the meeting!
```

PHP addAttribute() 函数

定义和用法

addAttribute() 函数给 SimpleXML 元素添加一个属性。

该函数无返回值。

语法

```
class SimpleXMLElement
{
string addAttribute(name,value,ns)
}
```

参数	描述
name	必需。规定属性的名称。
value	必需。规定属性的值。
ns	可选。规定属性的命名空间。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php
$xml = simplexml_load_file("test.xml");

$xml->body[0]->addAttribute("type", "small");

foreach($xml->body[0]->attributes() as $a => $b)
  {
  echo $a,'="',$b,'"';
  }
?>
```

输出：

```
type="small"
```

PHP addChild() 函数

定义和用法

addChild() 函数向指定的 XML 节点添加一个子节点。

该函数返回一个 SimpleXMLElement 对象，这个对象表示添加到 XML 节点的子元素。

语法

```
class SimpleXMLElement
{
string addChild(name,value,ns)
}
```

参数	描述
name	必需。规定子元素的名称。
value	必需。规定子元素的值。
ns	可选。规定子元素的命名空间。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php
$xml = simplexml_load_file("test.xml");

$xml->body[0]->addChild("date", "2008-08-08");

foreach ($xml->body->children() as $child)
  {
  echo "Child node: " . $child;
  }
?>
```

输出：

```
Child node: 2008-08-08
```

PHP asXML() 函数

定义和用法

asXML() 函数以字符串的形式从 SimpleXMLElement 对象返回 XML 文档。

若失败，则返回 false。

语法

```
class SimpleXMLElement
{
string asXML(file)
}
```

参数	描述
file	可选。如果规定了此参数，则该函数会把 XML 写入一个文件，而不是返回它。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php
if (file_exists('test.xml'))
  {
  $xml = simplexml_load_file('test.xml');
  }

echo $xml->asXML();
?>
```

输出：

```
George John Reminder Don't forget the meeting!
```

如果在浏览器窗口中选择“查看源文件”，会看到这些 HTML：

```
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP attributes() 函数

定义和用法

attributes() 函数获取 SimpleXML 元素的属性。

该函数提供在一个 XML 标签中定义的属性和值。

语法

```
class SimpleXMLElement
{
string attributes(_ns_,_is_prefix_)
}
```

参数	描述
ns	可选。被检索的属性的命名空间。
is_prefix	可选。默认是 false。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body type="small" important="low">Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php
$xml = simplexml_load_file("test.xml");

foreach($xml->body[0]->attributes() as $a => $b)
  {
  echo $a,'="',$b,'"';
  }
?>
```

输出：

```
type="small" important="low"
```

PHP children() 函数

定义和用法

children() 函数获取指定节点的子节点。

语法

```
class SimpleXMLElement
{
string children(ns,is_prefix)
}
```

参数	描述
ns	可选。
is_prefix	可选。默认是 false。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php
$xml = simplexml_load_file("test.xml");

foreach ($xml->children() as $child)
  {
  echo "Child node: " . $child;
  }
?>
```

输出类似：

```
Child node: George
Child node: John
Child node: Reminder
Child node: Don't forget the meeting!
```

PHP getDocNamespaces() 函数

定义和用法

getDocNamespaces() 函数从 SimpleXMLElement 对象返回在 XML 文档中声明的命名空间。

如果成功，该函数返回包含命名空间名称（带有关联的 URL）的数组。如果失败，则返回 false。

语法

```
class SimpleXMLElement
{
string getDocNamespaces(recursive)
}
```

参数	描述
recursive	可选。规定是否返回父子节点中的所有命名空间。默认是 false。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note xmlns:b="http://www.w3school.com.cn/example/">
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<b:body>Don't forget the meeting!</b:body>
</note>
```

PHP 代码：

```
<?php
if (file_exists('test.xml'))
  {
  $xml = simplexml_load_file('test.xml');
  }

print_r($xml->getDocNamespaces());
?>
```

输出类似：

```
Array
(
[b] => http://www.w3school.com.cn/example/
)
```

PHP getName() 函数

定义和用法

getName() 函数从 SimpleXMLElement 对象获取 XML 元素的名称。

如果成功，该函数返回当前的 XML 元素的名称。如果失败，则返回 false。

语法

```
class SimpleXMLElement
{
string getName()
}
```

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<b:body>Don't forget the meeting!</b:body>
</note>
```

PHP 代码：

```
<?php
if (file_exists('test.xml'))
  {
  $xml = simplexml_load_file('test.xml');
  }

echo $xml->getName();

foreach($xml->children() as $child)
  {
  echo $child->getName();
  }
?>
```

输出类似：

```
note
to
from
heading
body
```

PHP getNamespace() 函数

定义和用法

getNamespace() 函数获取在 XML 文档中使用的命名空间。

如果成功，该函数返回命名空间（带有关联的 URL）的一个数组。如果失败，则返回 false。

语法

```
class SimpleXMLElement
{
string getNamespace(recursive)
}
```

参数	描述
recursive	可选。规定是否返回父子节点中使用的所有命名空间。默认是 false。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note xmlns:b="http://www.w3school.com.cn/example/">
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<b:body>Don't forget the meeting!</b:body>
</note>
```

PHP 代码：

```
<?php
if (file_exists('test.xml'))
  {
  $xml = simplexml_load_file('test.xml');
  }

print_r($xml->getNamespaces());
?>
```

输出类似：

```
Array
(
[b] => http://www.w3school.com.cn/example/
)
```

PHP registerXPathNamespace() 函数

定义和用法

registerXPathNamespace() 函数为下一次 XPath 查询创建命名空间语境。

语法

```
class SimpleXMLElement
{
string registerXPathNamespace(prefix,ns)
}
```

参数	描述
prefix	必需。规定命名空间前缀。
ns	必需。规定命名空间 URL。必须匹配 XML 文档中的命名空间。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note xmlns:b="http://www.w3school.com.cn/example/">
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<b:body>Don't forget the meeting!</b:body>
</note>
```

PHP 代码：

```
<?php
$xml = simplexml_load_file("test.xml");

$xml->registerXPathNamespace("msg", "http://www.w3school.com.cn/example/");
$result = $xml->xpath("msg:body");

foreach ($result as $message)
  {
  echo $message;
  }
?>
```

输出类似：

```
Don't forget the meeting!
```

PHP simplexml_import_dom() 函数

定义和用法

simplexml_import_dom() 函数把 DOM 节点转换为 SimpleXMLElement 对象。

如果失败，则该函数返回 false。

语法

```
simplexml_import_dom(data,class)
```

参数	描述
data	必需。规定要使用的 DOM 节点。
class	必需。规定新对象的 class。

例子

```
<?php
$dom = new domDocument;
$dom->loadXML('<note><from>John</from></note>');

$xml = simplexml_import_dom($dom);

echo $xml->from;
?>
```

输出类似：

```
John
```

PHP simplexml_load_file() 函数

定义和用法

simplexml_load_file() 函数把 XML 文档载入对象中。

如果失败，则返回 false。

语法

```
simplexml_load_file(file,class,options,ns,is_prefix)
```

参数	描述
file	必需。规定要使用的 XML 文档。
class	可选。规定新对象的 class。
options	可选。规定附加的 Libxml 参数。
ns	可选。
is_prefix	可选。

返回值

返回类 SimpleXMLElement 的一个对象，该对象的属性包含 XML 文档中的数据。如果失败，则返回 false。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php
if (file_exists('test.xml'))
  {
  $xml = simplexml_load_file('test.xml');
  var_dump($xml);
  }

else
  {
  exit('Error.');
  }
?>
```

输出：

```
object(SimpleXMLElement)#1 (4)
{
["to"]=> string(4) "George"
["from"]=> string(4) "John"
["heading"]=> string(8) "Reminder"
["body"]=> string(29) "Don't forget the meeting!"
}
```

PHP simplexml_load_string() 函数

定义和用法

simplexml_load_string() 函数把 XML 字符串载入对象中。

如果失败，则返回 false。

语法

```
simplexml_load_file(string,class,options,ns,is_prefix)
```

参数	描述
string	必需。规定要使用的 XML 字符串。
class	可选。规定新对象的 class。
options	可选。规定附加的 Libxml 参数。
ns	可选。
is_prefix	可选。

返回值

返回类 SimpleXMLElement 的一个对象，该对象的属性包含 XML 文档中的数据。如果失败，则返回 false。

例子

```
<?php
$xmlstring = <<<XML
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
XML;

$xml = simplexml_load_string($xmlstring);

var_dump($xml);
?>
```

输出：

```
object(SimpleXMLElement)#1 (4)
{
["to"]=> string(4) "George"
["from"]=> string(4) "John"
["heading"]=> string(8) "Reminder"
["body"]=> string(29) "Don't forget the meeting!"
}
```

PHP xpath() 函数

定义和用法

xpath() 函数运行对 XML 文档的 XPath 查询。

如果成功，则返回包含 SimpleXMLElement 对象的一个数组。如果失败，则返回 false。

语法

```
class SimpleXMLElement
{
string xpath(path)
}
```

参数	描述
path	必需。XPath 路径。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php
$xml = simplexml_load_file("test.xml");

$result = $xml->xpath("from");

print_r($result);
?>
```

输出：

```
Array
(
[0] => SimpleXMLElement Object
  (
  [0] => John
  )
)
```

PHP String 函数

PHP String 简介

String 字符串函数允许您对字符串进行操作。

安装

String 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP String 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
addcslashes()	在指定的字符前添加反斜杠。	4
addslashes()	在指定的预定义字符前添加反斜杠。	3
bin2hex()	把 ASCII 字符的字符串转换为十六进制值。	3
chop()	rtrim() 的别名。	3
chr()	从指定的 ASCII 值返回字符。	3
chunk_split()	把字符串分割为一连串更小的部分。	3
convert_cyr_string()	把字符由一种 Cyrillic 字符转换成另一种。	3
convert_uudecode()	对 uuencode 编码的字符串进行解码。	5
convert_uuencode()	使用 uuencode 算法对字符串进行编码。	5
count_chars()	返回字符串所用字符的信息。	4
crc32()	计算一个字符串的 32-bit CRC。	4
crypt()	单向的字符串加密法 (hashing)。	3
echo()	输出字符串。	3
explode()	把字符串打散为数组。	3
fprintf()	把格式化的字符串写到指定的输出流。	5
get_html_translation_table()	返回翻译表。	4
hebrev()	把希伯来文本从右至左的流转换为左至右的流。	3

hebrevc()	同上，同时把(\n) 转为 。	3
html_entity_decode()	把 HTML 实体转换为字符。	4
htmlentities()	把字符转换为 HTML 实体。	3
htmlspecialchars_decode()	把一些预定义的 HTML 实体转换为字符。	5
htmlspecialchars()	把一些预定义的字符转换为 HTML 实体。	3
implode()	把数组元素组合为一个字符串。	3
join()	implode() 的别名。	3
levenshtein()	返回两个字符串之间的 Levenshtein 距离。	3
localeconv()	返回包含本地数字及货币信息格式的数组。	4
ltrim()	从字符串左侧删除空格或其他预定义字符。	3
md5()	计算字符串的 MD5 散列。	3
md5_file()	计算文件的 MD5 散列。	4
metaphone()	计算字符串的 metaphone 键。	4
money_format()	把字符串格式化为货币字符串。	4
nl_langinfo()	返回指定的本地信息。	4
nl2br()	在字符串中的每个新行之前插入 HTML 换行符。	3
number_format()	通过千位分组来格式化数字。	3
ord()	返回字符串第一个字符的 ASCII 值。	3
parse_str()	把查询字符串解析到变量中。	3
print()	输出一个或多个字符串。	3
printf()	输出格式化的字符串。	3
quoted_printable_decode()	解码 quoted-printable 字符串。	3
quotemeta()	在字符串中某些预定义的字符前添加反斜杠。	3
rtrim()	从字符串的末端开始删除空白字符或其他预定义字符。	3
setlocale()	设置地区信息（地域信息）。	3
sha1()	计算字符串的 SHA-1 散列。	4
sha1_file()	计算文件的 SHA-1 散列。	4
similar_text()	计算两个字符串的匹配字符的数目。	3
soundex()	计算字符串的 soundex 键。	3
sprintf()	把格式化的字符串写写入一个变量中。	3
sscanf()	根据指定的格式解析来自一个字符串的输入。	4

str_ireplace()	替换字符串中的一些字符。（对大小写不敏感）	5
str_pad()	把字符串填充为新的长度。	4
str_repeat()	把字符串重复指定的次数。	4
str_replace()	替换字符串中的一些字符。（对大小写敏感）	3
str_rot13()	对字符串执行 ROT13 编码。	4
str_shuffle()	随机地打乱字符串中的所有字符。	4
str_split()	把字符串分割到数组中。	5
str_word_count()	计算字符串中的单词数。	4
strcasecmp()	比较两个字符串。（对大小写不敏感）	3
strchr()	搜索字符串在另一字符串中的第一次出现。strstr() 的别名	3
strcmp()	比较两个字符串。（对大小写敏感）	3
strcoll()	比较两个字符串（根据本地设置）。	4
strcspn()	返回在找到任何指定的字符之前，在字符串查找的字符数。	3
strip_tags()	剥去 HTML、XML 以及 PHP 的标签。	3
stripcslashes()	删除由 addcslashes() 函数添加的反斜杠。	4
stripslashes()	删除由 addslashes() 函数添加的反斜杠。	3
stripos()	返回字符串在另一字符串中第一次出现的位置(大小写不敏感)	5
stristr()	查找字符串在另一字符串中第一次出现的位置(大小写不敏感)	3
strlen()	返回字符串的长度。	3
strnatcasecmp()	使用一种“自然”算法来比较两个字符串（对大小写不敏感）	4
strnatcmp()	使用一种“自然”算法来比较两个字符串（对大小写敏感）	4
strncasecmp()	前 n 个字符的字符串比较（对大小写不敏感）。	4
strncmp()	前 n 个字符的字符串比较（对大小写敏感）。	4
strpbrk()	在字符串中搜索指定字符中的任意一个。	5
strpos()	返回字符串在另一字符串中首次出现的位置（对大小写敏感）	3
strrchr()	查找字符串在另一个字符串中最后一次出现的位置。	3
strrev()	反转字符串。	3

strripos()	查找字符串在另一字符串中最后出现的位置(对大小写不敏感)	5
strrpos()	查找字符串在另一字符串中最后出现的位置(对大小写敏感)	3
strspn()	返回在字符串中包含的特定字符的数目。	3
strstr()	搜索字符串在另一字符串中的首次出现（对大小写敏感）	3
strtok()	把字符串分割为更小的字符串。	3
strtolower()	把字符串转换为小写。	3
strtoupper()	把字符串转换为大写。	3
strtr()	转换字符串中特定的字符。	3
substr()	返回字符串的一部分。	3
substr_compare()	从指定的开始长度比较两个字符串。	5
substr_count()	计算子串在字符串中出现的次数。	4
substr_replace()	把字符串的一部分替换为另一个字符串。	4
trim()	从字符串的两端删除空白字符和其他预定义字符。	3
ucfirst()	把字符串中的首字符转换为大写。	3
ucwords()	把字符串中每个单词的首字符转换为大写。	3
vfprintf()	把格式化的字符串写到指定的输出流。	5
vprintf()	输出格式化的字符串。	4
vsprintf()	把格式化字符串写入变量中。	4
wordwrap()	按照指定长度对字符串进行折行处理。	4

PHP String 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
CRYPT_SALT_LENGTH	包含系统默认加密方法的长度。对于标准 DES 加密，长度是 2。	
CRYPT_STD_DES	如果支持 2 字符 salt 的 DES 加密，则设置为 1，否则为 0。	
CRYPT_EXT_DES	如果支持 9 字符 salt 的 DES 加密，则设置为 1，否则为 0。	
CRYPT_MD5	如果支持以1开始的 12 字符 salt 的MD5加密，则设置为1，否则为0。	
CRYPT_BLOWFISH	如果支持以 2 或 $2a$ 开始的 16 字符 salt 的 Blowfish 加密，则设置为 1，否则为 0。	
HTML_SPECIALCHARS		
HTML_ENTITIES		
ENT_COMPAT		
ENT_QUOTES		
ENT_NOQUOTES		
CHAR_MAX		
LC_CTYPE		
LC_NUMERIC		
LC_TIME		
LC_COLLATE		
LC_MONETARY		
LC_ALL		
LC_MESSAGES		
STR_PAD_LEFT		
STR_PAD_RIGHT		
STR_PAD_BOTH		

PHP addcslashes() 函数

定义和用法

addcslashes() 函数在指定的字符前添加反斜杠。

语法

```
addcslashes(_string_,_characters_)
```

参数	描述
string	必需。规定要检查的字符串。
characters	可选。规定受 addcslashes() 影响的字符或字符范围。

提示和注释

注释：在对 0，r，n 和 t 应用 addcslashes() 时要小心。在 PHP 中，\0，\r，\n 和 \t 是预定义的转义序列。

实例

例子 1

在本例中，我们要向字符串中的特定字符添加反斜杠：

```
<?php
$str = "Hello, my name is John Adams.";
echo $str;
echo addcslashes($str,'m');
echo addcslashes($str,'J');
?>
```

输出：

```
Hello, my name is John Adams.
Hello, \my na\me is John Ada\ms.
Hello, my name is \John Adams.
```

例子 2

在本例中，我们要向字符串中的一个范围内的字符添加反斜杠：

```
<?php
$str = "Hello, my name is John Adams.";
echo $str;
echo addcslashes($str,'A..Z');
echo addcslashes($str,'a..z');
echo addcslashes($str,'a..h');
?>
```

输出：

```
Hello, my name is John Adams.
\Hello, my name is \John \Adams.
H\e\l\l\o, \m\y \n\a\m\e \i\s J\o\h\n A\d\a\m\s.
H\ello, my n\am\e is Jo\hn A\d\ams.
```

PHP addslashes() 函数

定义和用法

addslashes() 函数在指定的预定义字符前添加反斜杠。

这些预定义字符是：

- 单引号 (')
- 双引号 (")
- 反斜杠 ()
- NULL

语法

```
addslashes(string)
```

参数	描述
string	必需。规定要检查的字符串。

提示和注释

提示：该函数可用于为存储在数据库中的字符串以及数据库查询语句准备合适的字符串。

注释：默认情况下，PHP 指令 magic_quotes_gpc 为 on，对所有的 GET、POST 和 COOKIE 数据自动运行 addslashes()。不要对已经被 magic_quotes_gpc 转义过的字符串使用 addslashes()，因为这样会导致双层转义。遇到这种情况时可以使用函数 get_magic_quotes_gpc() 进行检测。

例子

在本例中，我们要向字符串中的预定义字符添加反斜杠：

```
<?php
$str = "Who's John Adams?";
echo $str . " This is not safe in a database query.<br />";
echo addslashes($str) . " This is safe in a database query.";
?>
```

输出：

```
Who's John Adams? This is not safe in a database query.
Who\'s John Adams? This is safe in a database query.
```

PHP bin2hex() 函数

定义和用法

bin2hex() 函数把 ASCII 字符的字符串转换为十六进制值。

语法

```
bin2hex(string)
```

参数	描述
string	必需。规定要转换的字符串。

例子

在本例中，我们将把一个字符串值从二进制转换为十六进制，再转换回去：

```
<?php
$str = "Hello world!";
echo bin2hex($str);
echo pack("H*",bin2hex($str));
?>
```

输出：

```
48656c6c6f20776f726c6421
Hello world!
```

PHP chop() 函数

定义和用法

chop() 函数从字符串的末端开始删除空白字符或其他预定义字符。

该函数的 rtrim() 函数的别名。

语法

```
chop(string,charlist)
```

参数	描述
string	必需。规定要转换的字符串。
charlist	可选。规定从字符串中删除哪些字符。如果未设置该参数，则全部删除以下字符："\0" - ASCII 0, NULL "\t" - ASCII 9, 制表符 "\n" - ASCII 10, 新行 "\x0B" - ASCII 11, 垂直制表符 "\r" - ASCII 13, 回车 " " - ASCII 32, 空格

例子

在本例中，我们将使用 chop() 函数从字符串右端删除字符：

```
<?php
$str = "Hello World!\n\n";
echo $str;
echo chop($str);
?>
```

以上代码输出的源代码：

```
<html>

<body>
Hello World!

Hello World!</body>

</html>
```

输出：

```
Hello World! Hello World!
```

PHP chr() 函数

定义和用法

chr() 函数从指定的 ASCII 值返回字符。

语法

```
chr(ascii)
```

参数	描述
ascii	必需。ASCII 值。

提示和注释

注释：ascii 参数可以是十进制、八进制或十六进制。通过前置 0 来规定八进制，通过前置 0x 来规定十六进制。

例子

```
<?php
echo chr(52);
echo chr(052);
echo chr(0x52);
?>
```

输出：

```
4
*
R
```

PHP chunk_split() 函数

定义和用法

chunk_split() 函数把字符串分割为一连串更小的部分。

语法

```
chunk_split(string,length,end)
```

参数	描述
string	必需。规定要分割的字符串。
length	可选。一个数字，定义字符串块的长度。
end	可选。字符串值，定义在每个字符串块之后放置的内容。

提示和注释

注释：本函数不改变原始字符串。

例子

例子 1

本例分隔每个字符，并添加 "."：

```
<?php
$str = "Hello world!";
echo chunk_split($str,1,".");
?>
```

输出：

```
H.e.l.l.o. .w.o.r.l.d.!.
```

例子 2

本例将在六个字符之后分割一次字符串，并添加 "..."：

```
<?php
$str = "Hello world!";
echo chunk_split($str,6,"...");
?>
```

输出：

```
Hello ...world!...
```

PHP convert_cyr_string() 函数

定义和用法

convert_cyr_string() 函数把字符由一种 Cyrillic 字符转换成另一种。

被支持的 Cyrillic 字符集是：

- k - koi8-r
- w - windows-1251
- i - iso8859-5
- a - x-cp866
- d - x-cp866
- m - x-mac-cyrillic

语法

```
convert_cyr_string(string,from,to)
```

参数	描述
string	必需。规定要转换的字符串。
from	必需。源 Cyrillic 字符集。
to	必需。目标 Cyrillic 字符集。

提示和注释

注释：本函数可安全用于二进制对象。

PHP convert_uudecode() 函数

定义和用法

convert_uudecode() 函数对 uuencode 编码的字符串进行解码。

语法

```
convert_uudecode(string)
```

参数	描述
string	必需。规定要解码的字符串。

例子

在本例中，我们将通过使用 convert_uudecode() 对 uuencode 编码的字符串进行解码：

```
<?php
$str = ",2&5L;&\@=V]R;&0A `";
echo convert_uudecode($str);
?>
```

输出：

```
Hello world!
```

PHP convert_uuencode() 函数

定义和用法

convert_uuencode() 函数使用 uuencode 算法对字符串进行编码。

语法

```
convert_uuencode(string)
```

参数	描述
string	必需。规定进行 uuencode 的字符串。

提示和注释

注释：本函数把所有字符串（包括二进制的）转换为可打印的字符串，确保其网络传输的安全。

注释：uuencode 的字符串比原字符串增大大约 35%。

例子

在本例中，我们将使用 convert_uuencode() 对字符串进行编码：

```
<?php
$str = "Hello world!";
echo convert_uuencode($str);
?>
```

输出：

```
,2&5L;&\@=V]R;&0A `
```

PHP count_chars() 函数

定义和用法

count_chars() 函数返回字符串所用字符的信息。

语法

```
count_chars(_string_,_mode_)
```

参数	描述
string	必需。规定要检查的字符串。
mode	可选。规定返回模式。默认是 0。有以下不同的返回模式： `0` - 数组，ASCII 值为键名，出现的次数为键值 `1` - 数组，ASCII 值为键名，出现的次数为键值，只列出出现次数大于 0 的值 `2` - 数组，ASCII 值为键名，出现的次数为键值，只列出出现次数等于 0 的值 `3` - 字符串，带有所有使用过的不同的字符 `4` - 字符串，带有所有未使用过的不同的字符

实例

例子 1

在本例中，我们将使用 count_chars() 来检查字符串，返回模式设置为 1：

```
<?php
$str = "Hello World!";
print_r(count_chars($str,1));
?>
```

输出：

```
Array
(
[32] => 1
[33] => 1
[72] => 1
[87] => 1
[100] => 1
[101] => 1
[108] => 3
[111] => 2
[114] => 1
)
```

例子 2

在本例中，我们将使用 count_chars() 来检查字符串，返回模式设置为 3：

```
<?php
$str = "Hello World!";
echo count_chars($str,3);
?>
```

输出：

```
 !HWdelor
```

PHP crc32() 函数

定义和用法

crc32() 函数计算一个字符串的 crc32 多项式。

该函数可用于验证数据的完整性。

语法

```
crc32(string)
```

参数	描述
string	必需。规定要计算的字符串。

说明

生成 string 参数的 32 位循环冗余校验码多项式。这通常用于检查传输的数据是否完整。

提示和注释

提示：由于 PHP 的整数是带符号的，许多 crc32 校验码将返回负整数，因此您需要使用 sprintf() 或 printf() 的 "%u" 格式符来获取表示无符号 crc32 校验码的字符串。

例子

例子 1

在本例中，我们将在使用以及不使用 "%u" 格式符的情况下，输出 crc32() 的结果（注意结果是相同的）：

```
<?php
$str = crc32("Hello world!");
echo 'Without %u: '.$str."<br />";
echo 'With %u: ';
printf("%u",$str);
?>
```

输出：

```
Without %u: 461707669
With %u: 461707669
```

例子 2

在本例中，我们将在使用以及不使用 "%u" 格式符的情况下，输出 crc32() 的结果（注意结果是不相同的）：

```
<?php
$str = crc32("Hello world.");
echo 'Without %u: '.$str."<br />";
echo 'With %u: ';
printf("%u",$str);
?>
```

输出：

```
Without %u: -1959132156
With %u: 2335835140
```

PHP crypt() 函数

定义和用法

crypt() 函数返回使用 DES、Blowfish 或 MD5 加密的字符串。

在不同的操作系统上，本函数的行为不同，某些操作系统支持一种以上的算法类型。在安装时，PHP 会检查什么算法可用以及使用什么算法。

语法

```
crypt(str,salt)
```

参数	描述
str	必需。规定要编码的字符串。
salt	可选。用于增加被编码字符数目的字符串，以使编码更加安全。如果未提供 salt 参数，则每次调用该函数时会随机生成一个。

说明

确切的算法依赖于 salt 参数的格式和长度。

下面是与 crypt() 函数一起使用的一些常量。在安装时，由 PHP 设置这些常量：

- [CRYPT_SALT_LENGTH]
- [CRYPT_STD_DES]
- [CRYPT_EXT_DES]
- [CRYPT_MD5]
- [CRYPT_BLOWFISH]

提示和注释

提示：解密函数是没有的。crypt() 函数使用一种单向算法。

例子

在本例中，我们将测试不同的算法：

```
<?php
if (CRYPT_STD_DES == 1)
{
echo "Standard DES: ".crypt("hello world")."\n<br />";
}
else
{
echo "Standard DES not supported.\n<br />";
}

if (CRYPT_EXT_DES == 1)
{
echo "Extended DES: ".crypt("hello world")."\n<br />";
}
else
{
echo "Extended DES not supported.\n<br />";
}

if (CRYPT_MD5 == 1)
{
echo "MD5: ".crypt("hello world")."\n<br />";
}
else
{
echo "MD5 not supported.\n<br />";
}

if (CRYPT_BLOWFISH == 1)
{
echo "Blowfish: ".crypt("hello world");
}
else
{
echo "Blowfish DES not supported.";
}
?>
```

输出类似（依赖于操作系统）：

```
Standard DES: $1$r35.Y52.$iyiFuvM.zFGsscpU0aZ4e.
Extended DES not supported.
MD5: $1$BN1.0I2.$8oBI/4mufxK6Tq89M12mk/
Blowfish DES not supported.
```

PHP echo() 函数

定义和用法

echo() 函数输出一个或多个字符串。

语法

```
echo(strings)
```

参数	描述
strings	必需。一个或多个要发送到输出的字符串。

提示和注释

注释：echo() 实际上不是一个函数，因此您无需对其使用括号。不过，如果您希望向 echo() 传递一个或多个参数，那么使用括号会发生解析错误。

提示：echo() 函数比 print() 函数快一点点。

提示：echo() 函数可以使用简化语法。参见例子 5。

例子

例子 1

```
<?php
$str = "Who's John Adams?";
echo $str;
echo "<br />";
echo $str."<br />I don't know!";
?>
```

输出：

```
Who's John Adam?
Who's John Adam?
I don't know!
```

例子 2

```
<?php
echo "This text
spans multiple
lines.";
?>
```

输出：

```
This text spans multiple lines.
```

例子 3

```
<?php
echo 'This ','string ','was ','made ','with multiple parameters';
?>
```

输出：

```
This string was made with multiple parameters
```

例子 4

单引号和双引号的不同之处。单引号仅输出变量名，而不是值：

```
<?php
$color = "red";
echo "Roses are $color";
echo "<br />";
echo 'Roses are $color';
?>
```

输出：

```
Roses are red
Roses are $color
```

例子 5

简化语法：

```
<html>
<body>

<?php
$color = "red";
?>

<p>Roses are <?=$color?></p>

</body>
</html>
```

PHP explode() 函数

定义和用法

explode() 函数把字符串分割为数组。

语法

```
explode(separator,string,limit)
```

参数	描述
separator	必需。规定在哪里分割字符串。
string	必需。要分割的字符串。
limit	可选。规定所返回的数组元素的最大数目。

说明

本函数返回由字符串组成的数组，其中的每个元素都是由 *separator* 作为边界点分割出来的子字符串。

separator 参数不能是空字符串。如果 *separator* 为空字符串（""），explode() 将返回 FALSE。如果 *separator* 所包含的值在 *string* 中找不到，那么 explode() 将返回包含 *string* 中单个元素的数组。

如果设置了 *limit* 参数，则返回的数组包含最多 *limit* 个元素，而最后那个元素将包含 *string* 的剩余部分。

如果 *limit* 参数是负数，则返回除了最后的 *-limit* 个元素外的所有元素。此特性是 PHP 5.1.0 中新增的。

提示和注释

注释：参数 limit 是在 PHP 4.0.1 中加入的。

注释：由于历史原因，虽然 implode() 可以接收两种参数顺序，但是 explode() 不行。你必须保证 *separator* 参数在 *string* 参数之前才行。

例子

在本例中，我们将把字符串分割为数组：

```
<?php
$str = "Hello world. It's a beautiful day.";
print_r (explode(" ",$str));
?>
```

输出：

```
Array
(
[0] => Hello
[1] => world.
[2] => It's
[3] => a
[4] => beautiful
[5] => day.
)
```

PHP fprintf() 函数

定义和用法

fprintf() 函数把格式化的字符串写到指定的输出流（例如：文件或数据库）。

该函数返回被写字符串的长度。

语法

```
fprintf(stream,format,arg1,arg2,arg++)
```

参数	描述
stream	可选。规定在哪里写/输出字符串。
format	必需。转换格式。
arg1	必需。规定插到 format 字符串中第一个 % 符号处的参数。
arg2	可选。规定插到 format 字符串中第二个 % 符号处的参数。
arg++	可选。规定插到 format 字符串中第三、四等等 % 符号处的参数。

说明

参数 *format* 是转换的格式，以百分比符号 ("%") 开始到转换字符结束。下面的可能的 *format* 值：

- %% - 返回百分比符号
- %b - 二进制数
- %c - 依照 ASCII 值的字符
- %d - 带符号十进制数
- %e - 可续计数法（比如 1.5e+3）
- %u - 无符号十进制数
- %f - 浮点数(local settings aware)
- %F - 浮点数(not local settings aware)
- %o - 八进制数
- %s - 字符串
- %x - 十六进制数（小写字母）
- %X - 十六进制数（大写字母）

arg1, arg2, ++ 等参数将插入到主字符串中的百分号 (%) 符号处。该函数是逐步执行的。在第一个 % 符号中，插入 arg1，在第二个 % 符号处，插入 arg2，依此类推。

提示和注释

注释：如果 % 符号多于 arg 参数，则您必须使用占位符。占位符被插入 % 符号之后，由数字和 "$" 组成。请参见例子 3。

提示： 相关函数： printf()、 sprintf()、 vfprintf()、 vprintf() 以及 vsprintf()。

例子

例子 1

```
<?php
$str = "Hello";
$number = 123;
$file = fopen("test.txt","w");
echo fprintf($file,"%s world. Day number %u",$str,$number);
?>
```

输出：

```
27
```

以下文本将写入 "test.txt"：

```
Hello world. Day number 123
```

例子 2

```
<?php
$number = 123;
$file = fopen("test.txt","w");
fprintf($file,"%f",$number);
?>
```

输出：

```
123.000000
```

例子 3

使用占位符：

```
<?php
$number = 123;
$file = fopen("test.txt","w");
fprintf($file,"With 2 decimals: %1\$.2f\nWith no decimals: %1\$u",$number);
?>
```

以下文本将写入 "test.txt"：

```
With 2 decimals: 123.00
With no decimals: 123
```

PHP get_html_translation_table() 函数

定义和用法

get_html_translation_table() 函数返回被 htmlentities() 和 htmlspecialchars() 函数使用的翻译表。

语法

```
get_html_translation_table(function,quotestyle)
```

参数	描述
function	可选。规定返回哪个翻译表。默认是 HTML_SPECIALCHARS。可能的值： `HTML_ENTITIES` - 翻译所有需要 URL 编码的字符，以便正确地显示在网页上。 `HTML_SPECIALCHARS` - 翻译某些需要 URL 编码的字符，以便正确地显示在网页上。
salt	可选。定义如何对单引号和双引号进行编码。默认是 ENT_COMPAT。可能的值： `ENT_COMPAT` - 编码双引号，不编码单引号。 `ENT_QUOTES` - 编码双引号和单引号。 `ENT_NOQUOTES` - 不编码单引号或双引号。

说明

提示和注释

提示：一些字符可以按照若干种方式进行编码。get_html_translation_table() 返回最普通的编码。

例子

在本例中，我们将展示两种翻译表：

```
<?php
print_r (get_html_translation_table());
echo "<br />";
print_r (get_html_translation_table(HTML_ENTITIES));
?>
```

输出：

```
Array
(
["] => " [<] => < [>] => > [&] => &
)
Array
(
[ ] =>   [¡] => ¡ [¢] => ¢ [£] => £
[¤] => ¤ [¥] => ¥ [¦] => ¦ [§] => §
[¨] => ¨ [©] => © [ª] => ª [«] => «
[¬] => ¬ [] =>  [®] => ® [¯] => ¯
[°] => ° [±] => ± [²] => ² [³] => ³
[´] => ´ [µ] => µ [¶] => ¶ [·] => ·
[¸] => ¸ [¹] => ¹ [º] => º [»] => »
[¼] => ¼ [½] => ½ [¾] => ¾ [¿] => ¿
[À] => À [Á] => Á [Â] => Â [Ã] => Ã
[Ä] => Ä [Å] => Å [Æ] => Æ [Ç] => Ç
[È] => È [É] => É [Ê] => Ê [Ë] => Ë
[Ì] => Ì [Í] => Í [Î] => Î [Ï] => Ï
[Ð] => Ð [Ñ] => Ñ [Ò] => Ò [Ó] => Ó
[Ô] => Ô [Õ] => Õ [Ö] => Ö [×] => ×
[Ø] => Ø [Ù] => Ù [Ú] => Ú [Û] => Û
[Ü] => Ü [Ý] => Ý [Þ] => Þ [ß] => ß
[à] => à [á] => á [â] => â [ã] => ã
[ä] => ä [å] => å [æ] => æ [ç] => ç
[è] => è [é] => é [ê] => ê [ë] => ë
[ì] => ì [í] => í [î] => î [ï] => ï
[ð] => ð [ñ] => ñ [ò] => ò [ó] => ó
[ô] => ô [õ] => õ [ö] => ö [÷] => ÷
[ø] => ø [ù] => ù [ú] => ú [û] => û
[ü] => ü [ý] => ý [þ] => þ [ÿ] => ÿ
["] => " [<] => < [>] => > [&] => &
)
```

PHP hebrev() 函数

定义和用法

hebrev() 函数把希伯来文本从右至左的流转换为左至右的流。

只有 224 至 251 之间的 ASCII 字符，以及标点符号受到影响。

语法

```
hebrev(string,maxcharline)
```

参数	描述
string	必需。希伯来文本。
salt	规定每行的最大字符数。如果可能，hebrev() 将避免把单词断开。

提示和注释

提示：hebrev() 和 hebrevc() 可以把希伯来逻辑文本转换为希伯来可见文本。希伯来可见文本不需要特殊的右至左字符支持，这使它对于在 web 上显示希伯来文本很有用处。

PHP hebrevc() 函数

定义和用法

hebrevc() 函数把希伯来文本从右至左的流转换为左至右的流。它也会把新行 (\n) 转换为
。

只有 224 至 251 之间的 ASCII 字符，以及标点符号受到影响。

语法

```
hebrev(string,maxcharline)
```

参数	描述
string	必需。希伯来文本。
salt	规定每行的最大字符数。如果可能，hebrev() 将避免把单词断开。

提示和注释

提示：hebrev() 和 hebrevc() 可以把希伯来逻辑文本转换为希伯来可见文本。希伯来可见文本不需要特殊的右至左字符支持，这使它对于在 web 上显示希伯来文本很有用处。

PHP html_entity_decode() 函数

定义和用法

html_entity_decode() 函数把 HTML 实体转换为字符。

html_entity_decode() 是 htmlentities() 的反函数。

语法

```
html_entity_decode(string,quotestyle,character-set)
```

参数	描述
string	必需。规定要解码的字符串。
quotestyle	可选。规定如何解码单引号和双引号。 ENT_COMPAT - 默认。仅解码双引号。 ENT_QUOTES - 解码双引号和单引号。 ENT_NOQUOTES - 不解码任何引号。
character-set	可选。字符串值，规定要使用的字符集。 ISO-8859-1 - 默认。西欧。 ISO-8859-15 - 西欧（增加 Euro 符号以及法语、芬兰语字母）。 UTF-8 - ASCII 兼容多字节 8 比特 Unicode cp866 - DOS 专用 Cyrillic 字符集 cp1251 - Windows 专用 Cyrillic 字符集 cp1252 - Windows 专用西欧字符集 KOI8-R - 俄语 GB2312 - 简体中文，国家标准字符集 BIG5 - 繁体中文 BIG5-HKSCS - Big5 香港扩展 Shift_JIS - 日语 EUC-JP - 日语

提示和注释

提示：无法被识别的字符集将被忽略，并由 ISO-8859-1 代替。

例子

```
<?php
$str = "John &amp; &#039;Adams&#039;";
echo html_entity_decode($str);
echo "<br />";
echo html_entity_decode($str, ENT_QUOTES);
echo "<br />";
echo html_entity_decode($str, ENT_NOQUOTES);
?>
```

浏览器输出：

```
John & 'Adams'
John & 'Adams'
John & 'Adams'
```

如果在浏览器中查看源代码，会看到这些 HTML：

```
<html>
<body>
John & &#039;Adams&#039;<br />
John & 'Adams'<br />
John & &#039;Adams&#039;
</body>
</html>
```

PHP htmlentities() 函数

定义和用法

htmlentities() 函数把字符转换为 HTML 实体。

语法

```
htmlentities(string,quotestyle,character-set)
```

参数	描述
string	必需。规定要转换的字符串。
quotestyle	可选。规定如何编码单引号和双引号。 `ENT_COMPAT` - 默认。仅编码双引号。 `ENT_QUOTES` - 编码双引号和单引号。 `ENT_NOQUOTES` - 不编码任何引号。
character-set	可选。字符串值，规定要使用的字符集。 `ISO-8859-1` - 默认。西欧。 `ISO-8859-15` - 西欧（增加 Euro 符号以及法语、芬兰语字母）。 `UTF-8` - ASCII 兼容多字节 8 比特 Unicode `cp866` - DOS 专用 Cyrillic 字符集 `cp1251` - Windows 专用 Cyrillic 字符集 `cp1252` - Windows 专用西欧字符集 `KOI8-R` - 俄语 `GB2312` - 简体中文，国家标准字符集 `BIG5` - 繁体中文 `BIG5-HKSCS` - Big5 香港扩展 `Shift_JIS` - 日语 `EUC` -JP - 日语

提示和注释

提示：无法被识别的字符集将被忽略，并由 ISO-8859-1 代替。

例子

```
<html>
<body>
<?php
$str = "John & 'Adams'";
echo htmlentities($str, ENT_COMPAT);
echo "<br />";
echo htmlentities($str, ENT_QUOTES);
echo "<br />";
echo htmlentities($str, ENT_NOQUOTES);
?>
</body>
</html>
```

浏览器输出：

```
John & 'Adams'
John & 'Adams'
John & 'Adams'
```

如果在浏览器中查看源代码，会看到这些 HTML：

```
<html>
<body>
John &amp; 'Adams'<br />
John &amp; &#039;Adams&#039;<br />
John &amp; 'Adams'
</body>
</html>
```

PHP htmlspecialchars_decode() 函数

定义和用法

htmlspecialchars_decode() 函数把一些预定义的 HTML 实体转换为字符。

会被解码的 HTML 实体是：

- & 成为 & （和号）
- " 成为 " （双引号）
- ' 成为 ' （单引号）
- < 成为 < （小于）
- > 成为 > （大于）

语法

```
htmlspecialchars_decode(string,quotestyle)
```

参数	描述
string	必需。规定要解码的字符串。
quotestyle	可选。规定如何解码单引号和双引号。 ENT_COMPAT - 默认。仅解码双引号。 ENT_QUOTES - 解码双引号和单引号。 ENT_NOQUOTES - 不解码任何引号。

例子

```
<?php
$str = "John &amp; &#039;Adams&#039;";
echo htmlspecialchars_decode($str);
echo "<br />";
echo htmlspecialchars_decode($str, ENT_QUOTES);
echo "<br />";
echo htmlspecialchars_decode($str, ENT_NOQUOTES);
?>
```

浏览器输出：

```
John & 'Adams'
John & 'Adams'
John & 'Adams'
```

如果在浏览器中查看源代码，会看到这些 HTML：

```
<html>
<body>
John & &#039;Adams&#039;<br />
John & 'Adams'<br />
John & &#039;Adams&#039;
</body>
</html>
```

PHP htmlspecialchars() 函数

定义和用法

htmlspecialchars() 函数把一些预定义的字符转换为 HTML 实体。

预定义的字符是：

- & （和号） 成为 &
- " （双引号） 成为 "
- ' （单引号） 成为 '
- < （小于） 成为 <
- > （大于） 成为 >

语法

```
htmlspecialchars(string,quotestyle,character-set)
```

参数	描述
string	必需。规定要转换的字符串。
quotestyle	可选。规定如何编码单引号和双引号。 `ENT_COMPAT` - 默认。仅编码双引号。 `ENT_QUOTES` - 编码双引号和单引号。 `ENT_NOQUOTES` - 不编码任何引号。
character-set	可选。字符串值，规定要使用的字符集。 `ISO-8859-1` - 默认。西欧。 `ISO-8859-15` - 西欧（增加 Euro 符号以及法语、芬兰语字母）。 `UTF-8` - ASCII 兼容多字节 8 比特 Unicode `cp866` - DOS 专用 Cyrillic 字符集 `cp1251` - Windows 专用 Cyrillic 字符集 `cp1252` - Windows 专用西欧字符集 `KOI8-R` - 俄语 `GB2312` - 简体中文，国家标准字符集 `BIG5` - 繁体中文 `BIG5-HKSCS` - Big5 香港扩展 `Shift_JIS` - 日语 `EUC-JP` - 日语

提示和注释

提示：无法被识别的字符集将被忽略，并由 ISO-8859-1 代替。

例子

```
<html>
<body>
<?php
$str = "John & 'Adams'";
echo htmlspecialchars($str, ENT_COMPAT);
echo "<br />";
echo htmlspecialchars($str, ENT_QUOTES);
echo "<br />";
echo htmlspecialchars($str, ENT_NOQUOTES);
?>
</body>
</html>
```

浏览器输出：

```
John & 'Adams'
John & 'Adams'
John & 'Adams'
```

如果在浏览器中查看源代码，会看到这些 HTML：

```
<html>
<body>
John &amp; 'Adams'<br />
John &amp; &#039;Adams&#039;<br />
John &amp; 'Adams'
</body>
</html>
```

PHP implode() 函数

定义和用法

implode() 函数把数组元素组合为一个字符串。

语法

```
implode(separator,array)
```

参数	描述
separator	可选。规定数组元素之间放置的内容。默认是 ""（空字符串）。
array	必需。要结合为字符串的数组。

说明

虽然 *separator* 参数是可选的。但是为了向后兼容，推荐您使用使用两个参数。

提示和注释

注释：implode() 可以接收两种参数顺序。但是由于历史原因，explode() 是不行的。你必须保证 *separator* 参数在 *string* 参数之前才行。

例子

```
<?php
$arr = array('Hello','World!','Beautiful','Day!');
echo implode(" ",$arr);
?>
```

输出：

```
Hello World! Beautiful Day!
```

PHP join() 函数

定义和用法

join() 函数把数组元素组合为一个字符串。

join() 函数是 implode() 函数的别名。

语法

```
join(separator,array)
```

参数	描述
separator	可选。规定数组元素之间放置的内容。默认是 ""（空字符串）。
array	必需。要结合为字符串的数组。

说明

虽然 *separator* 参数是可选的。但是为了向后兼容，推荐您使用使用两个参数。

提示和注释

注释：join() 可以接收两种参数顺序。但是由于历史原因，explode() 是不行的。你必须保证 *separator* 参数在 *string* 参数之前才行。

例子

```
<?php
$arr = array('Hello','World!','Beautiful','Day!');
echo join(" ",$arr);
?>
```

输出：

```
Hello World! Beautiful Day!
```

PHP levenshtein() 函数

定义和用法

levenshtein() 函数返回两个字符串之间的 Levenshtein 距离。

Levenshtein 距离，又称编辑距离，指的是两个字符串之间，由一个转换成另一个所需的最少编辑操作次数。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。

例如把 kitten 转换为 sitting：

1. sitten （k→s）
2. sittin （e→i）
3. sitting （→g）

levenshtein() 函数给每个操作（替换、插入和删除）相同的权重。不过，您可以通过设置可选的 insert、replace、delete 参数，来定义每个操作的代价。

语法

```
levenshtein(string1,string2,insert,replace,delete)
```

参数	描述
string1	必需。要对比的第一个字符串。
string2	必需。要对比的第二个字符串。
insert	可选。插入一个字符的代价。默认是 1。
replace	可选。替换一个字符的代价。默认是 1。
delete	可选。删除一个字符的代价。默认是 1。

提示和注释

注释：如果其中一个字符串超过 255 个字符，levenshtein() 函数返回 -1。

注释：levenshtein() 函数对大小写不敏感。

注释：levenshtein() 函数比 similar_text() 函数更快。不过，similar_text() 函数提供需要更少修改的更精确的结果。

例子

```
<?php
echo levenshtein("Hello World","ello World");
echo "<br />";
echo levenshtein("Hello World","ello World",10,20,30);
?>
```

输出：

```
1
30
```

PHP localeconv() 函数

定义和用法

localeconv() 函数返回包含本地数字及货币信息格式的数组。

localeconv() 函数返回以下数组元素：

- [decimal_point] - 小数点字符
- [thousands_sep] - 千位分隔符
- [int_curr_symbol] - 货币符号 （例如：USD）
- [currency_symbol] - 货币符号 （例如：$）
- [mon_decimal_point] - 货币小数点符号
- [mon_thousands_sep] - 货币千位分隔符
- [positive_sign] - 正值符号
- [negative_sign] - 负值符号
- [int_frac_digits] - 国际小数数字
- [frac_digits] - 本地小数数字
- [p_cs_precedes] - if 如果货币符号在正值之前，则是 True (1)，否则是 False (0)。
- [p_sep_by_space] - True (1) 如果货币符号与正值之间有空间，则是 True (1)，否则是 False (0)。
- [n_cs_precedes] - True (1) if 货币符号在负值之前，则是 True (1)，否则是 False (0)。
- [p_sep_by_space] - True (1) 如果货币符号与负值之间有空间，则是 True (1)，否则是 False (0)。
- [p_sign_posn] - 格式化选项：
  - 0 - 在数量和货币符号周围的圆括号
  - 1 - 数量和货币符号之前的 + 号
  - 2 - 数量和货币符号之后的 + 号
  - 3 - 货币符号之前的 + 号
  - 4 - 货币符号之后的 + 号
- [n_sign_posn] - 格式化选项：
  - 0 - 在数量和货币符号周围的圆括号
  - 1 - 数量和货币符号之前的 - 号
  - 2 - 数量和货币符号之后的 - 号
  - 3 - 货币符号之前的 - 号
  - 4 - 货币符号之后的 - 号
- [grouping] - 显示如何分组数字的 Array（例如：3 指示 1 000 000）
- [mon_grouping] - 显示如何分组货币数字的 Array（例如：2 指示 1 00 00 00）

语法

```
localeconv()
```

提示和注释

提示：如需定义本地设置，请使用 setlocale() 函数。

例子

在本例中，我们将获得美国本地的数字格式化信息：

```
<?php
setlocale(LC_ALL, 'US');
$locale_info = localeconv();
print_r($locale_info);
?>
```

输出：

```
Array
(
[decimal_point] => .
[thousands_sep] => ,
[int_curr_symbol] => USD
[currency_symbol] => $
[mon_decimal_point] => .
[mon_thousands_sep] => ,
[positive_sign] =>
[negative_sign] => -
[int_frac_digits] => 2
[frac_digits] => 2
[p_cs_precedes] => 1
[p_sep_by_space] => 0
[n_cs_precedes] => 1
[n_sep_by_space] => 0
[p_sign_posn] => 3
[n_sign_posn] => 0
[grouping] => Array ([0] => 3)
[mon_grouping] => Array ([0] => 3)
)
```

PHP ltrim() 函数

定义和用法

ltrim() 函数从字符串左侧删除空格或其他预定义字符。

语法

```
ltrim(string,charlist)
```

参数	描述
string	必需。规定要转换的字符串。
charlist	可选。规定从字符串中删除哪些字符。如果未设置该参数，则全部删除以下字符： " `\0` " - ASCII 0, NULL " `\t` " - ASCII 9, 制表符 " `\n` " - ASCII 10, 新行 " `\x0B` " - ASCII 11, 垂直制表符 " `\r` " - ASCII 13, 回车 " " - ASCII 32, 空格

例子

例子 1

```
<html>
<body>
<?php
$str = "    Hello World!";
echo "Without ltrim: " . $str;
echo "<br />";
echo "With ltrim: " . ltrim($str);
?>
<body>
<html>
```

输出：

```
Without ltrim: Hello World!
With ltrim: Hello World!
```

如果在浏览器中查看源代码，会看到以下 HTML：

```
<html>
<body>
Without ltrim:     Hello World!<br />With ltrim: Hello World!
</body>
</html>
```

例子 2

```
<?php
$str = "\r\nHello World!";
echo "Without ltrim: " . $str;
echo "<br />";
echo "With ltrim: " . ltrim($str);
?>
```

输出：

```
Without ltrim: Hello World!
With ltrim: Hello World!
```

如果在浏览器中查看源代码，会看到以下 HTML：

```
<html>
<body>
Without ltrim:
Hello World!<br />With ltrim: Hello World!
</body>
</html>
```

PHP md5() 函数

定义和用法

md5() 函数计算字符串的 MD5 散列。

md5() 函数使用 RSA 数据安全，包括 MD5 报文摘译算法。

如果成功，则返回所计算的 MD5 散列，如果失败，则返回 false。

语法

```
md5(_string_,_raw_)
```

参数	描述
string	必需。规定要计算的字符串。
raw	可选。规定十六进制或二进制输出格式： `TRUE` - 原始 16 字符二进制格式 `FALSE` - 默认。32 字符十六进制数 注释：该参数是 PHP 5.0 中添加的。

例子

例子 1

```
<?php
$str = "Hello";
echo md5($str);
?>
```

输出：

```
8b1a9953c4611296a827abf8c47804d7
```

例子 2

```
<?php
$str = "Hello";
echo md5($str);

if (md5($str) == '8b1a9953c4611296a827abf8c47804d7')
  {
  echo "<br />Hello world!";
  exit;
  }
?>
```

输出：

```
8b1a9953c4611296a827abf8c47804d7
Hello world!
```

PHP md5_file() 函数

定义和用法

md5_file() 函数计算文件的 MD5 散列。

md5() 函数使用 RSA 数据安全，包括 MD5 报文摘译算法。

如果成功，则返回所计算的 MD5 散列，如果失败，则返回 false。

语法

```
md5(_string_,_raw_)
```

参数	描述
string	必需。规定要计算的文件。
raw	可选。规定十六进制或二进制输出格式： `TRUE` - 原始 16 字符二进制格式 `FALSE` - 默认。32 字符十六进制数注释：该参数是 PHP 5.0 中添加的。

例子

例子 1

```
<?php
$filename = "test.txt";
$md5file = md5_file($filename);
echo $md5file;
?>
```

输出：

```
5d41402abc4b2a76b9719d911017c592
```

例子 2

存储 "test.txt" 文件的 MD5 散列：

```
<?php
$md5file = md5_file("test.txt");
file_put_contents("md5file.txt",$md5file);
?>
```

在本例中，我们将检测 "test.txt" 是否已被更改（即是否 MD5 散列已被更改）：

```
<?php
$md5file = file_get_contents("md5file.txt");
if (md5_file("test.txt") == $md5file)
  {
  echo "The file is ok.";
  }
else
  {
  echo "The file has been changed.";
  }
?>
```

输出：

```
The file is ok.
```

定义和用法

metaphone() 函数计算字符串的 metaphone 键。

metaphone 键字符串的英语发音。

metaphone() 函数可用于拼写检查应用程序。

如果成功，则返回字符串的 metaphone 键，如果失败，则返回 false。

语法

```
metaphone(string,length)
```

参数	描述
string	必需。规定要检查的字符串。
length	可选。规定 metaphone 键的最大长度。

提示和注释

注释：metaphone() 为发音相似的单词创建相同的键。

注释：所生成的 metaphone 键长度可变。

提示：metaphone() 比 soundex() 函数更精确，因为 metaphone() 了解基本的英语发音规则。

例子

例子 1

```
<?php
echo metaphone("world");
?>
```

输出：

```
WRLT
```

例子 2

在本例中，我们对两个发音相似的单词应用 metaphone() 函数：

```
<?php
$str = "Sun";
$str2 = "Son";

echo metaphone($str);
echo metaphone($str2);
?>
```

输出：

```
SN
SN
```

PHP money_format() 函数

定义和用法

money_format() 函数把字符串格式化为货币字符串。

语法

```
money_format(string,number)
```

参数	描述
string	必需。规定要格式化的字符串。
number	可选。被插入格式化字符串中 % 符号位置的数字。

提示和注释

注释：money_format() 函数无法在 windows 平台上工作。

例子

例子 1

国际 en_US 格式：

```
<?php
$number = 1234.56;
setlocale(LC_MONETARY, "en_US");
echo money_format("The price is %i", $number);
?>
```

输出：

```
The price is USD 1,234.56
```

例子 2

负数，带有 () 指示负数的 US 国际格式，右侧精度为 2，"*" 为填充字符：

```
<?php
$number = -1234.5672;

echo money_format("%=*(#10.2n", $number);
?>
```

输出：

```
($********1,234.57)
```

PHP nl_langinfo() 函数

定义和用法

nl_langinfo() 函数返回指定的本地信息。

如果成功，则返回指定的本地信息。如果失败，则返回 false。

语法

```
nl_langinfo(element)
```

参数	描述
element	必需。规定要返回哪个元素。必须是说明中列出的元素之一。

说明

时间和日历：

- ABDAY_(1-7) - Abbreviated name of the numbered day of the week
- DAY_(1-7) - Name of the numbered day of the week (DAY_1 = Sunday)
- ABMON_(1-12) - Abbreviated name of the numbered month of the year
- MON_(1-12) - Name of the numbered month of the year
- AM_STR - String for Ante meridian
- PM_STR - String for Post meridian
- D_T_FMT - String that can be used as the format string for strftime() to represent time and date
- D_FMT - String that can be used as the format string for strftime() to represent date
- T_FMT - String that can be used as the format string for strftime() to represent time
- T_FMT_AMPM - String that can be used as the format string for strftime() to represent time in 12-hour format with ante/post meridian
- ERA - Alternate era
- ERA_YEAR - Year in alternate era format
- ERA_D_T_FMT - Date and time in alternate era format (string can be used in strftime())
- ERA_D_FMT - Date in alternate era format (string can be used in strftime())
- ERA_T_FMT - Time in alternate era format (string can be used in strftime())

货币类别：

- INT_CURR_SYMBOL - Currency symbol (example: USD)
- CURRENCY_SYMBOL - Currency symbol (example: $)
- CRNCYSTR - Same as CURRENCY_SYMBOL
- MON_DECIMAL_POINT - Monetary decimal point character
- MON_THOUSANDS_SEP - Monetary thousands separator
- POSITIVE_SIGN - Positive value character
- NEGATIVE_SIGN -Negative value character
- MON_GROUPING - Array displaying how monetary numbers are grouped (example: 1 000 000)
- INT_FRAC_DIGITS - International fractional digits
- FRAC_DIGITS - Local fractional digits
- P_CS_PRECEDES - True (1) if currency symbol is placed in front of a positive value, False (0) if it is placed behind
- P_SEP_BY_SPACE - True (1) if there is a spaces between the currency symbol and a positive value, False (0) otherwise
- N_CS_PRECEDES - True (1) if currency symbol is placed in front of a negative value, False (0) if it is placed behind
- N_SEP_BY_SPACE - True (1) if there is a spaces between the currency symbol and a negative value, False (0) otherwise
- P_SIGN_POSN - Formatting setting. Possible return values:
  - 0 - Parentheses surround the quantity and currency symbol
  - 1 - The sign string is placed in front of the quantity and currency symbol
  - 2 - The sign string is placed after the quantity and currency symbol
  - 3 - The sign string is placed immediately in front of the currency symbol
  - 4 - The sign string is placed immediately after the currency symbol
- N_SIGN_POSN - Formatting setting. Possible return values:
  - 0 - Parentheses surround the quantity and currency symbol
  - 1 - The sign string is placed in front of the quantity and currency symbol
  - 2 - The sign string is placed after the quantity and currency symbol
  - 3 - The sign string is placed immediately in front of the currency symbol
  - 4 - The sign string is placed immediately after the currency symbol

数字类别：

- DECIMAL_POINT - Decimal point character
- RADIXCHAR - Same as DECIMAL_POINT
- THOUSANDS_SEP - Separator character for thousands
- THOUSEP - Same as THOUSANDS_SEP
- GROUPING - Array displaying how numbers are grouped (example: 1 000 000)

通信类别：

- YESEXPR - Regex string for matching 'yes' input
- NOEXPR - Regex string for matching 'no' input
- YESSTR - Output string for 'yes'
- NOSTR - Output string for 'no'

代码集类别：

- CODESET Return a string with the name of the character encoding.

提示和注释

注释：money_format() 函数无法在 windows 平台上工作。

提示：与返回所有本地格式化信息的 localeconv() 函数不同，nl_langinfo() 返回指定的信息。

PHP nl2br() 函数

定义和用法

nl2br() 函数在字符串中的每个新行 (\n) 之前插入 HTML 换行符 (
)。

语法

```
nl2br(string)
```

参数	描述
string	必需。规定要检查的字符串。

例子

```
<?php
echo nl2br("One line.\nAnother line.");
?>
```

输出：

```
One line.
Another line.
```

HTML 代码：

```
One line.<br />
Another line.
```

PHP number_format() 函数

定义和用法

number_format() 函数通过千位分组来格式化数字。

语法

```
number_format(number,decimals,decimalpoint,separator)
```

参数	描述
number	必需。要格式化的数字。如果未设置其他参数，则数字会被格式化为不带小数点且以逗号 (,) 作为分隔符。
decimals	可选。规定多少个小数。如果设置了该参数，则使用点号 (.) 作为小数点来格式化数字。
decimalpoint	可选。规定用作小数点的字符串。
separator	可选。规定用作千位分隔符的字符串。仅使用该参数的第一个字符。比如 "xyz" 仅输出 "x"。注释：如果设置了该参数，那么所有其他参数都是必需的。

提示和注释

注释：该函数支持一个、两个或四个参数（不是三个）。

例子

```
<?php
echo number_format("1000000");
echo number_format("1000000",2);
echo number_format("1000000",2,",",".");
?>
```

输出：

```
1,000,000
1,000,000.00
1.000.000,00
```

PHP ord() 函数

定义和用法

ord() 函数返回字符串第一个字符的 ASCII 值。

语法

```
ord(string)
```

参数	描述
string	必需。要从中获得 ASCII 值的字符串。

例子

```
<?php
echo ord("h");
echo ord("hello");
?>
```

输出：

```
104
104
```

PHP parse_str() 函数

定义和用法

parse_str() 函数把查询字符串解析到变量中。

语法

```
parse_str(string,array)
```

参数	描述
string	必需。规定要解析的字符串。
array	可选。规定存储变量的数组名称。该参数指示变量存储到数组中。

提示和注释

注释：如果未设置 array 参数，由该函数设置的变量将覆盖已由同名变量。

注释：php.ini 中的 magic_quotes_gpc 设置影响该函数的输出。如果已启用，那么在 parse_str() 解析之前，变量会被 addslashes() 转换。

例子

例子 1

```
<?php
parse_str("id=23&name=John%20Adams");
echo $id."<br />";
echo $name;
?>
```

输出：

```
23
John Adams
```

例子 2

```
<?php
parse_str("id=23&name=John%20Adams",$myArray);
print_r($myArray);
?>
```

输出：

```
Array
(
[id] => 23
[name] => John Adams
)
```

PHP print() 函数

定义和用法

print() 函数输出一个或多个字符串。

语法

```
print(strings)
```

参数	描述
strings	必需。发送到输出的一个或多个字符串。

提示和注释

注释：print() 函数实际上不是函数，所以您不必对它使用括号。

注释：print() 函数稍慢于 echo()。

例子

例子 1

```
<?php
$str = "Who's John Adams?";
print $str;
print "<br />";
print $str."<br />I don't know!";
?>
```

输出：

```
Who's John Adams?
Who's John Adams?
I don't know!
```

例子 2

```
<?php
print "This text
spans multiple
lines.";
?>
```

输出：

```
This text spans multiple lines.
```

例子 3

```
<?php
$color = "red";
print "Roses are $color";
print "<br />";
print 'Roses are $color';
?>
```

输出：

```
Roses are red
Roses are $color
```

PHP printf() 函数

定义和用法

printf() 函数输出格式化的字符串。

语法

```
printf(format,arg1,arg2,arg++)
```

参数	描述
format	必需。规定字符串以及如何格式化其中的变量。
arg1	必需。规定插到格式化字符串中第一个 % 符号处的参数。
arg2	可选。规定插到格式化字符串中第二个 % 符号处的参数。
arg++	可选。规定插到格式化字符串中第三、四等等 % 符号处的参数。

说明

arg1, arg2, ++ 等参数将插入到主字符串中的百分号 (%) 符号处。该函数是逐步执行的。在第一个 % 符号中，插入 arg1，在第二个 % 符号处，插入 arg2，依此类推。

提示和注释

注释：如果 % 符号多于 arg 参数，则您必须使用占位符。占位符被插入 % 符号之后，由数字和 "$" 组成。请参见例子 3。

注释： 相关函数： fprintf()、 sprintf()、 vfprintf()、 vprintf() 以及 vsprintf()。

例子

例子 1

```
<?php
$str = "Hello";
$number = 123;
printf("%s world. Day number %u",$str,$number);
?>
```

输出：

```
Hello world. Day number 123
```

例子 2

```
<?php
$number = 123;
printf("%f",$number);
?>
```

输出：

```
123.000000
```

例子 3

使用占位符：

```
<?php
$number = 123;
printf("With 2 decimals: %1\$.2f<br />With no decimals: %1\$u",$number);
?>
```

输出：

```
With 2 decimals: 123.00
With no decimals: 123
```

PHP quoted_printable_decode() 函数

定义和用法

quoted_printable_decode() 函数对经过 quoted-printable 编码后的字符串进行解码，返回 8 位的字符串。

该函数类似于 imap_qprint() 函数。不同的是，应用 imap_qprint() 函数需要让系统加载 IMAP 模块，而本函数不需要加载 IMAP 模块。

语法

```
quoted_printable_decode(string)
```

参数	描述
string	必需。规定要解码的 quoted-printable 字符串。

例子

在本例中，应用 quoted_printable_decode() 函数对经过 quoted-printable 编码后的字符串 "=0A" 进行解码，并返回 8 位的字符串：

```
<?php
$str = "Hello=0Aworld.";
echo quoted_printable_decode($str);
?>
```

输出：

```
Hello world.
```

HTML 源代码：

```
Hello
world.
```

PHP quotemeta() 函数

定义和用法

quotemeta() 函数在字符串中某些预定义的字符前添加反斜杠。

这些预定义字符是：

- 句号 (.)
- 反斜杠 ()
- 加号 (+)
- 星号 (*)
- 问号 (?)
- 方括号 ([])
- 脱字符号 (^)
- 美元符号 ($)
- 圆括号 (())

语法

```
quotemeta(string)
```

参数	描述
string	必需。规定要检查的字符串。

提示和注释

提示：该函数可用于转义拥有特殊意义的字符，比如 SQL 中的 ( )、[ ] 以及 *。

例子

```
<?php
$str = "Hello world. (can you hear me?)";
echo quotemeta($str);
?>
```

输出：

```
Hello world\. \(can you hear me\?\)
```

PHP rtrim() 函数

定义和用法

chop() 函数从字符串的末端开始删除空白字符或其他预定义字符。

语法

```
rtrim(string,charlist)
```

参数	描述
string	必需。规定要转换的字符串。
charlist	可选。规定从字符串中删除哪些字符。如果未设置该参数，则全部删除以下字符：" `\0` " - ASCII 0, NULL " `\t` " - ASCII 9, 制表符 " `\n` " - ASCII 10, 新行 " `\x0B` " - ASCII 11, 垂直制表符 " `\r` " - ASCII 13, 回车 " " - ASCII 32, 空格

例子

在本例中，我们将使用 rtrim() 函数从字符串右端删除字符：

```
<?php
$str = "Hello World!\n\n";
echo $str;
echo rtrim($str);
?>
```

以上代码输出的源代码：

```
<html>

<body>
Hello World!

Hello World!</body>

</html>
```

输出：

```
Hello World! Hello World!
```

PHP setlocale() 函数

定义和用法

setlocale() 函数设置地区信息（地域信息）。

地区信息是针对一个地理区域的语言、货币、时间以及其他信息。

该函数返回当前的地区设置，若失败则返回 false。

语法

```
setlocale(constant,location)
```

参数	描述
constant	必需。规定应该设置什么地区信息。可用的常量： `LC_ALL` - 包括下面的所有选项 `LC_COLLATE` -排序次序 `LC_CTYPE` - 字符类别及转换（例如所有字符大写或小写） `LC_MESSAGES` - 系统消息格式 `LC_MONETARY` - 货币格式 `LC_NUMERIC` - 数字格式 `LC_TIME` - 日期/时间格式
location	必需。规定把地区信息设置为什么国家/地区。如果 location 参数是数组，setlocale() 会尝试每个数组元素，直到找到合法的语言或地区代码为止。如果某个地区在不同的系统上拥有不同的名称，这一点很有用。注释：在此查找语言和地区代码。

提示和注释

注释：setlocale() 函数仅针对当前脚本改变地区信息。

提示：可以通过 setlocale(LC_ALL,NULL) 把地区信息设置为系统默认。

例子

在本例中，我们将把 locale 设置为 US English，然后再设置回系统默认：

```
<?php
echo setlocale(LC_ALL,"En-Us");
echo setlocale(LC_ALL,NULL);
?>
```

PHP sha1() 函数

定义和用法

sha1() 函数计算字符串的 SHA-1 散列。

sha1() 函数使用美国 Secure Hash 算法 1。

如果成功，则返回所计算的 SHA-1 散列，如果失败，则返回 false。

语法

```
sha1(_string_,_raw_)
```

参数	描述
string	必需。规定要计算的字符串。
raw	可选。规定十六进制或二进制输出格式： `TRUE` - 原始 20 字符二进制格式 `FALSE` - 默认。40 字符十六进制数注释：该参数是 PHP 5.0 中添加的。

例子

例子 1

```
<?php
$str = 'Hello';
echo sha1($str);
?>
```

输出：

```
f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0
```

例子 2

在本例中，我们将输出 sha1() 的结果，然后对其测试：

```
<?php
$str = 'Hello';
echo sha1($str);

if (sha1($str) == 'f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0')
  {
  echo "<br />Hello world!";
  exit;
  }
?>
```

输出：

```
f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0
Hello world!
```

PHP sha1_file() 函数

定义和用法

sha1_file() 函数计算文件的 SHA-1 散列。

sha1() 函数使用美国 Secure Hash 算法 1。

如果成功，则返回所计算的 SHA-1 散列，如果失败，则返回 false。

语法

```
sha1_file(_string_,_raw_)
```

参数	描述
string	必需。规定要计算的文件。
raw	可选。规定十六进制或二进制输出格式： `TRUE` - 原始 20 字符二进制格式 `FALSE` - 默认。40 字符十六进制数注释：该参数是 PHP 5.0 中添加的。

例子

例子 1

```
<?php
$filename = "test.txt";
$sha1file = sha1_file($filename);
echo $sha1file;
?>
```

输出：

```
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
```

例子 2

在一个文件中存储 "test.txt" 的 SHA-1 散列：

```
<?php
$sha1file = sha1_file("test.txt");
file_put_contents("sha1file.txt",$sha1file);
?>
```

在本例中，我们将测试 "test.txt" 是否已更改（即 SHA-1 hash 是否已更改）：

```
<?php
$sha1file = file_get_contents("sha1file.txt");
if (sha1_file("test.txt") == $sha1file)
  {
  echo "The file is ok.";
  }
else
  {
  echo "The file has been changed.";
  }
?>
```

输出：

```
The file is ok.
```

PHP similar_text() 函数

定义和用法

similar_text() 函数计算两个字符串的匹配字符的数目。

该函数也可以计算两个字符串的相似度（以百分比计）。

语法

```
similar_text(string1,string2,percent)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。
percent	可选。规定供存储百分比相似度的变量名。

提示和注释

注释：levenshtein() 函数比 similar_text() 函数更快。不过，similar_text() 函数通过更少的必需修改次数提供更精确的结果。

例子

例子 1

```
<?php
echo similar_text("Hello World","Hello Peter");
?>
```

输出：

```
7
```

例子 2

```
<?php
similar_text("Hello World","Hello Peter",$percent);
echo $percent;
?>
```

输出：

```
63.6363636364
```

PHP soundex() 函数

定义和用法

soundex() 函数计算字符串的 soundex 键。

soundex 键是 4 字符长的字母数字字符串，表示一个单词的英文发音。

soundex() 函数可用于拼写检查程序。

如果成功，则返回字符串的 soundex 键，如果失败，则返回 false。

语法

```
soundex(string)
```

参数	描述
string	必需。规定要检查的字符串。

提示和注释

注释：soundex() 为发音相似的单词创建相同的键。

提示：metaphone() 比 soundex() 函数更精确，因为 metaphone() 了解基本的英语发音规则。

例子

例子 1

```
<?php
$str = "hello";
echo soundex($str);
?>
```

输出：

```
H400
```

例子 2

在本例中，我们对两个发音相似的单词应用 soundex() 函数：

```
<?php
$str = "Sun";
$str2 = "Son";

echo soundex($str);
echo "<br />";
echo soundex($str2);
?>
```

输出：

```
S500
S500
```

PHP sprintf() 函数

定义和用法

sprintf() 函数把格式化的字符串写入一个变量中。

语法

```
sprintf(_format_,_arg1_,_arg2_,_arg++_)
```

参数	描述
format	必需。转换格式。
arg1	必需。规定插到 *format* 字符串中第一个 % 符号处的参数。
arg2	可选。规定插到 *format* 字符串中第二个 % 符号处的参数。
arg++	可选。规定插到 *format* 字符串中第三、四等等 % 符号处的参数。

说明

参数 *format* 是转换的格式，以百分比符号 ("%") 开始到转换字符结束。下面的可能的 *format* 值：

- %% - 返回百分比符号
- %b - 二进制数
- %c - 依照 ASCII 值的字符
- %d - 带符号十进制数
- %e - 可续计数法（比如 1.5e+3）
- %u - 无符号十进制数
- %f - 浮点数(local settings aware)
- %F - 浮点数(not local settings aware)
- %o - 八进制数
- %s - 字符串
- %x - 十六进制数（小写字母）
- %X - 十六进制数（大写字母）

arg1, *arg2*, ++ 等参数将插入到主字符串中的百分号 (%) 符号处。该函数是逐步执行的。在第一个 % 符号中，插入 *arg1*，在第二个 % 符号处，插入 *arg2*，依此类推。

提示和注释

注释：如果 % 符号多于 *arg* 参数，则您必须使用占位符。占位符插到 % 符号后面，由数字和 "$" 组成。请参见例子 3。

提示： 相关函数： fprintf()、 printf()、 vfprintf()、 vprintf() 以及 vsprintf()。

例子

例子 1

```
<?php
$str = "Hello";
$number = 123;
$txt = sprintf("%s world. Day number %u",$str,$number);
echo $txt;
?>
```

输出：

```
Hello world. Day number 123
```

例子 2

```
<?php
$number = 123;
$txt = sprintf("%f",$number);
echo $txt;
?>
```

输出：

```
123.000000
```

例子 3

```
<?php
$number = 123;
$txt = sprintf("With 2 decimals: %1\$.2f<br />With no decimals: %1\$u",$number);
echo $txt;
?>
```

输出：

```
With 2 decimals: 123.00
With no decimals: 123
```

PHP sscanf() 函数

定义和用法

sscanf() 函数根据指定的格式解析来自一个字符串的输入。

如果只向该函数传递两个参数，数据将以数组的形式返回。否则，如果传递了额外的参数，那么被解析的数据会存储在这些参数中。如果区分符的数目大于包含它们的变量的数目，则会发生错误。不过，如果区分符少于变量，则额外的变量包含 NULL。

语法

```
sscanf(string,format,arg1,arg2,arg++)
```

参数	描述
string	必需。规定要读取的字符串。
format	必需。规定要使用的格式。
arg1	可选。存储数据的第一个变量。
arg2	可选。存储数据的第二个变量。
arg++	可选。存储数据的第三、四个变量。依此类推。

说明

参数 *format* 是转换的格式，以百分比符号 ("%") 开始到转换字符结束。下面的可能的 *format* 值：

- %% - 返回百分比符号
- %b - 二进制数
- %c - 依照 ASCII 值的字符
- %d - 带符号十进制数
- %e - 可续计数法（比如 1.5e+3）
- %u - 无符号十进制数
- %f - 浮点数(local settings aware)
- %F - 浮点数(not local settings aware)
- %o - 八进制数
- %s - 字符串
- %x - 十六进制数（小写字母）

- %X - 十六进制数（大写字母）

例子

```
<?php
$string = "age:30 weight:60kg";
sscanf($string,"age:%d weight:%dkg",$age,$weight);
// show types and values
var_dump($age,$weight);
?>
```

输出：

```
int(30)
int(60)
```

PHP str_ireplace() 函数

定义和用法

str_ireplace() 函数使用一个字符串替换字符串中的另一些字符。

语法

```
str_ireplace(find,replace,string,count)
```

参数	描述
find	必需。规定要查找的值。
replace	必需。规定替换 *find* 中的值的值。
string	必需。规定被搜索的字符串。
count	可选。一个变量，对替换数进行计数。

提示和注释

注释：该函数对大小写不敏感。请使用 str_replace() 执行对大小写敏感的搜索。

注释：该函数是二进制安全的。

例子

例子 1

```
<?php
echo str_ireplace("world","John","Hello world!");
?>
```

输出：

```
Hello John!
```

例子 2

在本例中，我们将演示带有数组和 count 变量的 str_ireplace() 函数：

```
<?php
$arr = array("blue","red","green","yellow");
print_r(str_ireplace("red","pink",$arr,$i));
echo "Replacements: $i";
?>
```

输出：

```
Array
(
[0] => blue
[1] => pink
[2] => green
[3] => yellow
)
Replacements: 1
```

例子 3

```
<?php
$find = array("Hello","world");
$replace = array("B");
$arr = array("Hello","world","!");
print_r(str_ireplace($find,$replace,$arr));
?>
```

输出：

```
Array
(
[0] => B
[1] =>
[2] => !
)
```

PHP str_pad() 函数

定义和用法

str_pad() 函数把字符串填充为指定的长度。

语法

```
str_pad(string,length,pad_string,pad_type)
```

参数	描述
string	必需。规定要填充的字符串。
length	必需。规定新字符串的长度。如果该值小于原始字符串的长度，则不进行任何操作。
pad_string	可选。规定供填充使用的字符串。默认是空白。
pad_type	可选。规定填充字符串的那边。可能的值： STR_PAD_BOTH - 填充到字符串的两头。如果不是偶数，则右侧获得额外的填充。 STR_PAD_LEFT - 填充到字符串的左侧。 STR_PAD_RIGHT - 填充到字符串的右侧。这是默认的。

例子

例子 1

```
<?php
$str = "Hello World";
echo str_pad($str,20,".");
?>
```

输出：

```
Hello World.........
```

例子 2

```
<?php
$str = "Hello World";
echo str_pad($str,20,".",STR_PAD_LEFT);
?>
```

输出：

```
.........Hello World
```

例子 3

```
<?php
$str = "Hello World";
echo str_pad($str,20,".:",STR_PAD_BOTH);
?>
```

输出：

```
.:.:Hello World.:.:.
```

PHP str_repeat() 函数

定义和用法

str_repeat() 函数把字符串重复指定的次数。

语法

```
str_repeat(string,repeat)
```

参数	描述
string	必需。规定要重复的字符串。
repeat	必需。规定字符串将被重复的次数。必须大于等于 0。

例子

```
<?php
echo str_repeat(".",13);
?>
```

输出：

```
.............
```

PHP str_replace() 函数

定义和用法

str_replace() 函数使用一个字符串替换字符串中的另一些字符。

语法

```
str_replace(find,replace,string,count)
```

参数	描述
find	必需。规定要查找的值。
replace	必需。规定替换 *find* 中的值的值。
string	必需。规定被搜索的字符串。
count	可选。一个变量，对替换数进行计数。

提示和注释

注释：该函数对大小写敏感。请使用 str_ireplace() 执行对大小写不敏感的搜索。

注释：该函数是二进制安全的。

例子

例子 1

```
<?php
echo str_replace("world","John","Hello world!");
?>
```

输出：

```
Hello John!
```

例子 2

在本例中，我们将演示带有数组和 count 变量的 str_replace() 函数：

```
<?php
$arr = array("blue","red","green","yellow");
print_r(str_replace("red","pink",$arr,$i));
echo "Replacements: $i";
?>
```

输出：

```
Array
(
[0] => blue
[1] => pink
[2] => green
[3] => yellow
)
Replacements: 1
```

例子 3

```
<?php
$find = array("Hello","world");
$replace = array("B");
$arr = array("Hello","world","!");
print_r(str_replace($find,$replace,$arr));
?>
```

输出：

```
Array
(
[0] => B
[1] =>
[2] => !
)
```

PHP str_rot13() 函数

定义和用法

str_rot13() 函数对字符串执行 ROT13 编码。

ROT-13 编码是一种每一个字母被另一个字母代替的方法。这个代替字母是由原来的字母向前移动 13 个字母而得到的。数字和非字母字符保持不变。

语法

```
str_rot13(string)
```

参数	描述
string	必需。规定要编码的字符串。

提示和注释

提示：编码和解码都是由相同的函数完成的。如果您把一个已编码的字符串作为参数，那么将返回原始字符串。

例子

在本例中，我们将通过使用 str_rot13() 函数对字符串进行编码和解码：

```
<?php
echo str_rot13("Hello World");
echo "<br />";
echo str_rot13("Uryyb Jbeyq");
?>
```

输出：

```
Uryyb Jbeyq
Hello World
```

PHP str_shuffle() 函数

定义和用法

str_shuffle() 函数随机地打乱字符串中的所有字符。

语法

```
str_shuffle(string)
```

参数	描述
string	必需。规定要打乱的字符串。

例子

```
<?php
echo str_shuffle("Hello World");
?>
```

输出：

```
H elooWlrdl
```

PHP str_split() 函数

定义和用法

str_split() 函数把字符串分割到数组中。

语法

```
str_split(string,length)
```

参数	描述
string	必需。规定要分割的字符串。
length	可选。规定每个数组元素的长度。默认是 1。

说明

如果 *length* 小于 1，str_split() 函数将返回 false。

如果 *length* 大于字符串的长度，整个字符串将作为数组的唯一元素返回。

例子

例子 1

```
<?php
print_r(str_split("Hello"));
?>
```

输出：

```
Array
(
[0] => H
[1] => e
[2] => l
[3] => l
[4] => o
)
```

例子 2

```
<?php
print_r(str_split("Hello",3));
?>
```

输出：

```
Array
(
[0] => Hel
[1] => lo
)
```

PHP str_word_count() 函数

定义和用法

str_word_count() 函数计算字符串中的单词数。

语法

```
str_word_count(string,return,char)
```

参数	描述
string	必需。规定要检查的字符串。
return	可选。规定 str_word_count() 函数的返回值。可能的值： 0 - 默认。返回找到的单词的数目。 1 - 返回包含字符串中的单词的数组。 2 - 返回一个数组，其中的键是单词在字符串中的位置，值是实际的单词。
return	可选。规定被认定为单词的特殊字符。该参数是 PHP 5.1 中新加的。

例子

例子 1

```
<?php
echo str_word_count("Hello world!");
?>
```

输出：

```
2
```

例子 2

```
<?php
print_r(str_word_count("Hello world!",1));
?>
```

输出：

```
Array
(
[0] => Hello
[1] => world
)
```

例子 3

```
<?php
print_r(str_word_count("Hello world!",2));
?>
```

输出：

```
Array
(
[0] => Hello
[6] => world
)
```

例子 4

```
<?php
print_r(str_word_count("Hello world & good morning!",1));
print_r(str_word_count("Hello world & good morning!",1,"&"));
?>
```

输出：

```
Array
(
[0] => Hello
[1] => world
[2] => good
[3] => morning
)

Array
(
[0] => Hello
[1] => world
[2] => &
[3] => good
[4] => morning
)
```

PHP strcasecmp() 函数

定义和用法

strcasecmp() 函数比较两个字符串。

该函数返回：

- 0 - 如果两个字符串相等
- <0 - 如果 string1 小于 string2
- >0 - 如果 string1 大于 string2

语法

```
strcasecmp(string1,string2)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。

提示和注释

注释：该函数是二进制安全的，且对大小写不敏感。

例子

```
<?php
echo strcasecmp("Hello world!","HELLO WORLD!");
?>
```

输出：

```
0
```

PHP strchr() 函数

定义和用法

strchr() 函数搜索一个字符串在另一个字符串中的第一次出现。

该函数返回字符串的其余部分（从匹配点）。如果未找到所搜索的字符串，则返回 false。

该函数是 strstr() 函数的别名。

语法

```
strchr(string,search)
```

参数	描述
string	必需。规定被搜索的字符串。
search	必需。规定所搜索的字符串。如果该参数是数字，则搜索匹配数字 ASCII 值的字符。

提示和注释

注释：该函数是二进制安全的。

注释：该函数对大小写敏感。如需进行大小写不敏感的搜索，请使用 stristr()。

例子

例子 1

```
<?php
echo strchr("Hello world!","world");
?>
```

输出：

```
world!
```

例子 2

在本例中，我们将搜索 "o" 的 ASCII 值所代表的字符：

```
<?php
echo strchr("Hello world!",111);
?>
```

输出：

```
o world!
```

PHP strcmp() 函数

定义和用法

strcmp() 函数比较两个字符串。

该函数返回：

- 0 - 如果两个字符串相等
- <0 - 如果 string1 小于 string2
- >0 - 如果 string1 大于 string2

语法

```
strcmp(string1,string2)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。

提示和注释

注释：该函数是二进制安全的，且对大小写敏感。

例子

```
<?php
echo strcmp("Hello world!","Hello world!");
?>
```

输出：

```
0
```

PHP strcoll() 函数

定义和用法

strcoll() 函数比较两个字符串。

该函数返回：

- 0 - 如果两个字符串相等
- <0 - 如果 string1 小于 string2
- >0 - 如果 string1 大于 string2

字符串的比较会根据本地设置而变化。（A<a 或 A>a）。

语法

```
strcoll(string1,string2)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。

提示和注释

注释：该函数对大小写敏感，但不是二进制安全的。

注释：如果本地设置是 C 或 POSIX，则该函数的工作方式与 strcmp() 相同。

例子

```
<?php
setlocale (LC_COLLATE, 'NL');
echo strcoll("Hello World!","Hello WORLD!");
echo "<br />";

setlocale (LC_COLLATE, 'en_US');
echo strcoll("Hello World!","Hello WORLD!");
?>
```

输出：

```
-1
1
```

PHP strcspn() 函数

定义和用法

strcspn() 函数返回在找到任何指定的字符之前，在字符串查找的字符数。

语法

```
strcspn(string,char,start,length)
```

参数	描述
string	必需。规定要搜索的字符串。
char	必需。规定要查找的字符。
start	可选。规定开始查找的位置。该参数是 PHP 4.3 中新加的。
length	可选。规定字符串的长度（搜索多少字符）。该参数是 PHP 4.3 中新加的。

提示和注释

注释：该函数是二进制安全的。

例子

```
<?php
echo strcspn("Hello world!","w");
?>
```

输出：

```
6
```

PHP strip_tags() 函数

定义和用法

strip_tags() 函数剥去 HTML、XML 以及 PHP 的标签。

语法

```
strip_tags(string,allow)
```

参数	描述
string	必需。规定要检查的字符串。
allow	可选。规定允许的标签。这些标签不会被删除。

提示和注释

注释：该函数始终会剥离 HTML 注释。这点无法通过 *allow* 参数改变。

例子

例子 1

```
<?php
echo strip_tags("Hello <b>world!</b>");
?>
```

输出：

```
Hello world!
```

例子 2

```
<?php
echo strip_tags("Hello <b><i>world!</i></b>","<b>");
?>
```

输出：

```
Hello **world!**
```

PHP stripcslashes() 函数

定义和用法

stripcslashes() 函数删除由 addcslashes() 函数添加的反斜杠。

语法

```
stripcslashes(string)
```

参数	描述
string	必需。规定要检查的字符串。

提示和注释

注释：该函数用于清理从数据库中取回的数据。

例子

```
<?php
echo stripcslashes("Hello, \my na\me is Kai Ji\m.");
?>
```

输出：

```
Hello, my name is Kai Jim.
```

PHP stripslashes() 函数

定义和用法

stripslashes() 函数删除由 addslashes() 函数添加的反斜杠。

语法

```
stripslashes(string)
```

参数	描述
string	必需。规定要检查的字符串。

提示和注释

注释：该函数用于清理从数据库或 HTML 表单中取回的数据。

例子

```
<?php
echo stripslashes("Who\'s John Adams?");
?>
```

输出：

```
Who's John Adams?
```

PHP stripos() 函数

定义和用法

stripos() 函数返回字符串在另一个字符串中第一次出现的位置。

如果没有找到该字符串，则返回 false。

语法

```
stripos(string,find,start)
```

参数	描述
string	必需。规定被搜索的字符串。
find	必需。规定要查找的字符。
start	可选。规定开始搜索的位置。

提示和注释

注释：该函数对大小写不敏感。如需进行对大小写敏感的搜索，请使用 strpos() 函数。

例子

```
<?php
echo stripos("Hello world!","WO");
?>
```

输出：

```
6
```

PHP stristr() 函数

定义和用法

stristr() 函数查找字符串在另一个字符串中第一次出现的位置。

如果成功，则返回字符串的其余部分（从匹配点）。如果没有找到该字符串，则返回 false。

语法

```
stristr(string,search)
```

参数	描述
string	必需。规定被搜索的字符串。
find	必需。规定要查找的字符。如果该参数是数字，则搜索匹配该数字对应的 ASCII 值的字符。

提示和注释

注释：该函数是二进制安全的。

注释：该函数对大小写不敏感。如需对大小写敏感的搜索，请使用 strstr()。

例子

例子 1

```
<?php
echo stristr("Hello world!","WORLD");
?>
```

输出：

```
world!
```

例子 2

```
<?php
echo stristr("Hello world!",111);
?>
```

输出：

```
o world!
```

PHP strlen() 函数

定义和用法

strlen() 函数返回字符串的长度。

语法

```
strlen(string)
```

参数	描述
string	必需。规定要检查的字符串。

例子

```
<?php
echo strlen("Hello world!");
?>
```

输出：

```
12
```

PHP strnatcasecmp() 函数

定义和用法

strnatcasecmp() 函数使用一种“自然”算法来比较两个字符串。

在自然算法中，数字 "2" 小于数字 "10"。在计算机排序中，"2" 大于 "10"，这是因为 "2" 大于 "10" 的第一个数字。

该函数返回：

- 0 - 如果两个字符串相等
- <0 - 如果 string1 小于 string2
- >0 - 如果 string1 大于 string2

语法

```
strnatcasecmp(string1,string2)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。

提示和注释

注释：该函数对大小写不敏感。

例子

```
<?php
echo strnatcasecmp("2Hello world!","10Hello world!");
echo "<br />";
echo strnatcasecmp("10Hello world!","2Hello world!");
?>
```

输出：

```
-1
1
```

PHP strnatcmp() 函数

定义和用法

strnatcmp() 函数使用一种“自然”算法来比较两个字符串。

在自然算法中，数字 "2" 小于数字 "10"。在计算机排序中，"2" 大于 "10"，这是因为 "2" 大于 "10" 的第一个数字。

该函数返回：

- 0 - 如果两个字符串相等
- <0 - 如果 string1 小于 string2
- >0 - 如果 string1 大于 string2

语法

```
strnatcmp(string1,string2)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。

提示和注释

注释：该函数对大小写敏感。

例子

```
<?php
echo strnatcmp("2Hello world!","10Hello world!");
echo "<br />";
echo strnatcmp("10Hello world!","2Hello world!");
?>
```

输出：

```
-1
1
```

PHP strncasecmp() 函数

定义和用法

strncasecmp() 函数比较两个字符串。

该函数返回：

- 0 - 如果两个字符串相等
- <0 - 如果 string1 小于 string2
- >0 - 如果 string1 大于 string2

语法

```
strncasecmp(string1,string2,length)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。
length	必需。规定每个字符串用于比较的字符数。

提示和注释

注释：该函数是二进制安全的，且对大小写不敏感。

例子

```
<?php
echo strncasecmp("Hello world!","Hello earth!",6);
?>
```

输出：

```
0
```

PHP strncmp() 函数

定义和用法

strncmp() 函数比较两个字符串。

该函数返回：

- 0 - 如果两个字符串相等
- <0 - 如果 string1 小于 string2
- >0 - 如果 string1 大于 string2

语法

```
strncmp(string1,string2,length)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。
length	必需。规定每个字符串用于比较的字符数。

提示和注释

注释：该函数是二进制安全的，且对大小写敏感。

例子

```
<?php
echo strncmp("Hello world!","Hello earth!",6);
?>
```

输出：

```
0
```

PHP strpbrk() 函数

定义和用法

strpbrk() 函数在字符串中搜索指定字符中的任意一个。

该函数返回指定字符第一次出现的位置开始的剩余部分。如果没有找到，则返回 false。

语法

```
strpbrk(string,charlist)
```

参数	描述
string	必需。规定被搜索的字符串。
charlist	必需。规定要查找的字符。

提示和注释

注释：该函数对大小写敏感。

例子

```
<?php
echo strpbrk("Hello world!","oe");
?>
```

输出：

```
ello world!
```

PHP strpos() 函数

定义和用法

strpos() 函数返回字符串在另一个字符串中第一次出现的位置。

如果没有找到该字符串，则返回 false。

语法

```
strpos(string,find,start)
```

参数	描述
string	必需。规定被搜索的字符串。
find	必需。规定要查找的字符。
start	可选。规定开始搜索的位置。

提示和注释

注释：该函数对大小写敏感。如需进行对大小写不敏感的搜索，请使用 stripos() 函数。

例子

```
<?php
echo strpos("Hello world!","wo");
?>
```

输出：

```
6
```

PHP strrchr() 函数

定义和用法

strrchr() 函数查找字符串在另一个字符串中最后一次出现的位置，并返回从该位置到字符串结尾的所有字符。

如果成失败，否则返回 false。

语法

```
strrchr(string,char)
```

参数	描述
string	必需。规定被搜索的字符串。
char	必需。规定要查找的字符。如果该参数是数字，则搜索匹配数字 ASCII 值的字符。

提示和注释

注释：该函数是二进制安全的。

例子

例子 1

```
<?php
echo strrchr("Hello world!","world");
?>
```

输出：

```
world!
```

例子 2

```
<?php
echo strrchr("Hello world!",111);
?>
```

输出：

```
orld!
```

PHP strrev() 函数

定义和用法

strrev() 函数反转字符串。

语法

```
strrev(string)
```

参数	描述
string	必需。规定要反转的字符串。

例子

```
<?php
echo strrev("Hello World!");
?>
```

输出：

```
!dlroW olleH
```

PHP strripos() 函数

定义和用法

strripos() 函数查找字符串在另一个字符串中最后一次出现的位置。

如果成功，则返回位置，否则返回 false。

语法

```
strrpos(string,find,start)
```

参数	描述
string	必需。规定被搜索的字符串。
find	必需。规定要查找的字符。
start	可选。规定开始搜索的位置。

提示和注释

注释：该函数对大小写不敏感。如需进行大小写敏感的查找，请使用 strrpos()。

例子

```
<?php
echo strripos("Hello world!","WO");
?>
```

输出：

```
6
```

PHP strrpos() 函数

定义和用法

strrpos() 函数查找字符串在另一个字符串中最后一次出现的位置。

如果成功，则返回位置，否则返回 false。

语法

```
strrpos(string,find,start)
```

参数	描述
string	必需。规定被搜索的字符串。
find	必需。规定要查找的字符。
start	可选。规定开始搜索的位置。

提示和注释

注释：该函数对大小写敏感。如需进行大小写不敏感的查找，请使用 strripos()。

例子

```
<?php
echo strrpos("Hello world!","wo");
?>
```

输出：

```
6
```

PHP strspn() 函数

定义和用法

strspn() 函数返回在字符串中包含的特定字符的数目。

语法

```
strspn(string,charlist,start,length)
```

参数	描述
string	必需。规定被搜索的字符串。
charlist	必需。规定要查找的字符。
start	可选。规定在字符串的何处开始。
length	可选。规定字符串的长度。

提示和注释

注释：该函数是二进制安全的。

例子

例子 1

```
<?php
echo strspn("Hello world!","kHlleo");
?>
```

输出：

```
5
```

例子 2

```
<?php
echo strspn("abcdefand","abc");
?>
```

输出：

```
3
```

PHP strstr() 函数

定义和用法

strstr() 函数搜索一个字符串在另一个字符串中的第一次出现。

该函数返回字符串的其余部分（从匹配点）。如果未找到所搜索的字符串，则返回 false。

语法

```
strstr(string,search)
```

参数	描述
string	必需。规定被搜索的字符串。
search	必需。规定所搜索的字符串。如果该参数是数字，则搜索匹配数字 ASCII 值的字符。

提示和注释

注释：该函数是二进制安全的。

注释：该函数对大小写敏感。如需进行大小写不敏感的搜索，请使用 stristr()。

例子

例子 1

```
<?php
echo strstr("Hello world!","world");
?>
```

输出：

```
world!
```

例子 2

在本例中，我们将搜索 "o" 的 ASCII 值所代表的字符：

```
<?php
echo strstr("Hello world!",111);
?>
```

输出：

```
o world!
```

PHP strtok() 函数

定义和用法

strtok() 函数把字符串分割为更小的字符串。

语法

```
strtok(string,split)
```

参数	描述
string	必需。规定要分割的字符串。
split	必需。规定一个或多个分割字符。

例子

在下面的例子中，请注意，我们仅在第一次调用 strtok() 函数时使用了 *string* 参数。在首次调用后，该函数仅需要 *split* 参数，这是因为它清楚自己在当前函数中所在的位置：

```
<?php
$string = "Hello world. Beautiful day today.";
$token = strtok($string, " ");

while ($token !== false)
  {
  echo "$token<br />";
  $token = strtok(" ");
  }
?>
```

输出：

```
Hello
world.
Beautiful
day
today.
```

PHP strtolower() 函数

定义和用法

strtolower() 函数把字符串转换为小写。

语法

```
strtolower(string)
```

参数	描述
string	必需。规定要转换的字符串。

例子

```
<?php
echo strtolower("Hello WORLD!");
?>
```

输出：

```
hello world!
```

PHP strtoupper() 函数

定义和用法

strtoupper() 函数把字符串转换为大写。

语法

```
strtoupper(string)
```

参数	描述
string	必需。规定要转换的字符串。

例子

```
<?php
echo strtoupper("Hello WORLD!");
?>
```

输出：

```
HELLO WORLD!
```

PHP strtr() 函数

定义和用法

strtr() 函数转换字符串中特定的字符。

语法

```
strtr(string,from,to)
```

或者

```
strtr(string,array)
```

参数	描述
string1	必需。规定要转换的字符串。
from	必需（除非使用数组）。规定要改变的字符。
to	必需（除非使用数组）。规定要改变为的字符。
array	必需（除非使用 *from* 和 *to*）。一个数组，其中的键是原始字符，值是目标字符。

说明

如果 *from* 和 *to* 的长度不同，则格式化为最短的长度。

例子

例子 1

```
<?php
echo strtr("Hilla Warld","ia","eo");
?>
```

输出：

```
Hello World
```

例子 2

```
<?php
$arr = array("Hello" => "Hi", "world" => "earth");
echo strtr("Hello world",$arr);
?>
```

输出：

```
Hi earth
```

PHP substr() 函数

定义和用法

substr() 函数返回字符串的一部分。

语法

```
substr(_string_,_start_,_length_)
```

参数	描述
string	必需。规定要返回其中一部分的字符串。
start	必需。规定在字符串的何处开始。 `正数` - 在字符串的指定位置开始 `负数` - 在从字符串结尾的指定位置开始 `0` - 在字符串中的第一个字符处开始
length	可选。规定要返回的字符串长度。默认是直到字符串的结尾。 `正数` - 从 start 参数所在的位置返回 `负数` - 从字符串末端返回

提示和注释

注释：如果 *start* 是负数且 *length* 小于等于 *start*，则 *length* 为 0。

例子

例子 1

```
<?php
echo substr("Hello world!",6);
?>
```

输出：

```
world!
```

例子 2

```
<?php
echo substr("Hello world!",6,5);
?>
```

输出：

```
world
```

PHP substr_compare() 函数

定义和用法

substr_compare() 函数从指定的开始长度比较两个字符串。

该函数返回：

- 0 - 如果两字符串相等
- <0 - 如果 string1 （从开始位置）小于 string2
- >0 - 如果 string1 （从开始位置）大于 string2

语法

```
substr_compare(string1,string2,startpos,length,case)
```

参数	描述
string1	必需。规定要比较的第一个字符串。
string2	必需。规定要比较的第二个字符串。
startpos	可选。规定在 *string1* 中的何处开始比较。
length	可选。规定在 *string1* 中参与比较的字符数。
case	可选。规定是否指定大小写比较。默认是 FALSE （对大小写敏感）。

说明

如果 *length* 大于或等于 *string1* 的长度，则函数返回 false。

例子

例子 1

```
<?php
echo substr_compare("Hello world","Hello world",0);
?>
```

输出：

```
0
```

例子 2

```
<?php
echo substr_compare("Hello world","world",6);
?>
```

输出：

```
0
```

例子 3

```
<?php
echo substr_compare("Hello world","WORLD",6,TRUE);
?>
```

输出：

```
0
```

PHP substr_count() 函数

定义和用法

substr_count() 函数计算子串在字符串中出现的次数。

语法

```
substr_count(_string_,_substring_,_start_,_length_)
```

参数	描述
string	必需。规定要检查的字符串。
substring	必需。规定要检索的字符串。
start	可选。规定在字符串中何处开始搜索。
length	可选。规定搜索的长度。

例子

```
<?php
echo substr_count("Hello world. The world is nice","world");
?>
```

输出：

```
2
```

PHP substr_replace() 函数

定义和用法

substr_replace() 函数把字符串的一部分替换为另一个字符串。

语法

```
substr_replace(string,replacement,start,length)
```

参数	描述
string	必需。规定要检查的字符串。
replacement	必需。规定要插入的字符串。
start	必需。规定在字符串的何处开始替换。 `正数` - 在第 *start* 个偏移量开始替换 `负数` - 在从字符串结尾的第 *start* 个偏移量开始替换 `0` - 在字符串中的第一个字符处开始替换
charlist	可选。规定要替换多少个字符。 `正数` - 被替换的字符串长度 `负数` - 从字符串末端开始的被替换字符数 `0` - 插入而非替换

提示和注释

注释：如果 *start* 是负数且 *length* 小于等于 *start*，则 *length* 为 0。

例子

```
<?php
echo substr_replace("Hello world","earth",6);
?>
```

输出：

```
Hello earth
```

PHP trim() 函数

定义和用法

trim() 函数从字符串的两端删除空白字符和其他预定义字符。

语法

```
trim(string,charlist)
```

参数	描述
string	必需。规定要检查的字符串。
charlist	可选。规定要转换的字符串。如果省略该参数，则删除以下所有字符：" \0 " - NULL " \t " - tab " \n " - new line " \x0B " - 纵向列表符 " \r " - 回车 " " - 普通空白字符

例子

例子 1

```
<html>
<body>
<?php
$str = "    Hello World!    ";
echo "Without trim: " . $str;
echo "<br />";
echo "With trim: " . trim($str);
?>
<body>
<html>
```

输出：

```
Without trim: Hello World!
With trim: Hello World!
```

HTML 源码：

```
<html>
<body>
Without trim:     Hello World!    <br />With trim: Hello World!
</body>
</html>
```

例子 2

```
<?php
$str = "\r\nHello World!\r\n";
echo "Without trim: " . $str;
echo "<br />";
echo "With trim: " . trim($str);
?>
```

输出：

```
Without trim: Hello World!
With trim: Hello World!
```

HTML 源码：

```
<html>
<body>
Without trim:
Hello World!
<br />With trim: Hello World!
</body>
</html>
```

PHP ucfirst() 函数

定义和用法

ucfirst() 函数把字符串中的首字符转换为大写。

语法

```
ucfirst(string)
```

参数	描述
string	必需。规定要转换的字符串。

例子

```
<?php
echo ucfirst("hello world");
?>
```

输出：

```
Hello world
```

PHP ucwords() 函数

定义和用法

ucwords() 函数把字符串中每个单词的首字符转换为大写。

语法

```
ucwords(string)
```

参数	描述
string	必需。规定要转换的字符串。

例子

```
<?php
echo ucwords("hello world");
?>
```

输出：

```
Hello World
```

PHP vfprintf() 函数

定义和用法

vfprintf() 函数把格式化的字符串写到指定的输出流。

与 fprintf() 不同，vfprintf() 中的 *arg* 参数位于数组中。数组的元素会被插入主字符串的百分比 (%) 符号处。该函数是逐步执行的。在第一个 % 符号中，插入 arg1，在第二个 % 符号处，插入 arg2，依此类推。

该函数返回被写的字符串的长度。

语法

```
vfprintf(stream,format,argarray)
```

参数	描述
stream	必需。规定在何处写/输出字符串。
format	必需。转换格式。
argarray	必需。带有参数的一个数组，这些参数会被插到 format 字符串中的 % 符号处。

说明

参数 *format* 是转换的格式，以百分比符号 ("%") 开始到转换字符结束。下面的可能的 *format* 值：

- %% - 返回百分比符号
- %b - 二进制数
- %c - 依照 ASCII 值的字符
- %d - 带符号十进制数
- %e - 可续计数法（比如 1.5e+3）
- %u - 无符号十进制数
- %f - 浮点数(local settings aware)
- %F - 浮点数(not local settings aware)
- %o - 八进制数
- %s - 字符串
- %x - 十六进制数（小写字母）

- %X - 十六进制数（大写字母）

提示和注释

注释：如果 % 符号多于 *arg* 参数，则您必须使用占位符。占位符插到 % 符号后面，由数字和 "$" 组成。请参见例子 3。

提示： 相关函数： fprintf()、 printf()、 sprintf()、 vprintf() 以及 vsprintf()。

例子

例子 1

```
<?php
$str = "Hello";
$number = 123;
$txt = sprintf("%s world. Day number %u",$str,$number);
echo $txt;
?>
```

输出：

```
Hello world. Day number 123
```

例子 2

```
<?php
$number = 123;
$txt = sprintf("%f",$number);
echo $txt;
?>
```

输出：

```
123.000000
```

例子 3

```
<?php
$number = 123;
$txt = sprintf("With 2 decimals: %1\$.2f<br />With no decimals: %1\$u",$number);
echo $txt;
?>
```

输出：

```
With 2 decimals: 123.00
With no decimals: 123
```

PHP vprintf() 函数

定义和用法

vprintf() 函数输出格式化的字符串。

与 printf() 不同，vprintf() 中的 *arg* 参数位于数组中。数组的元素会被插入主字符串的百分比 (%) 符号处。该函数是逐步执行的。在第一个 % 符号中，插入 arg1，在第二个 % 符号处，插入 arg2，依此类推。

语法

```
vprintf(format,argarray)
```

参数	描述
format	必需。转换格式。
argarray	必需。带有参数的一个数组，这些参数会被插到 format 字符串中的 % 符号处。

说明

参数 *format* 是转换的格式，以百分比符号 ("%") 开始到转换字符结束。下面的可能的 *format* 值：

- %% - 返回百分比符号
- %b - 二进制数
- %c - 依照 ASCII 值的字符
- %d - 带符号十进制数
- %e - 可续计数法（比如 1.5e+3）
- %u - 无符号十进制数
- %f - 浮点数(local settings aware)
- %F - 浮点数(not local settings aware)
- %o - 八进制数
- %s - 字符串
- %x - 十六进制数（小写字母）
- %X - 十六进制数（大写字母）

提示和注释

注释：如果 % 符号多于 *arg* 参数，则您必须使用占位符。占位符插到 % 符号后面，由数字和 "$" 组成。请参见例子 3。

提示： 相关函数： fprintf()、 printf()、 sprintf()、 vfprintf() 以及 vsprintf()。

例子

例子 1

```
<?php
$str = "Hello";
$number = 123;
vprintf("%s world. Day number %u",array($str,$number));
?>
```

输出：

```
Hello world. Day number 123
```

例子 2

```
<?php
$num1 = 123;
$num2 = 456;
vprintf("%f%f",array($num1,$num2));
?>
```

输出：

```
123.000000456.000000
```

例子 3

使用占位符：

```
<?php
$number = 123;
vprintf("With 2 decimals: %1\$.2f<br />With no decimals: %1\$u",array($number));
?>
```

输出：

```
With 2 decimals: 123.00
With no decimals: 123
```

PHP vsprintf() 函数

定义和用法

vsprintf() 函数把格式化字符串写入变量中。

与 sprintf() 不同，vsprintf() 中的 *arg* 参数位于数组中。数组的元素会被插入主字符串的百分比 (%) 符号处。该函数是逐步执行的。在第一个 % 符号中，插入 arg1，在第二个 % 符号处，插入 arg2，依此类推。

语法

```
vsprintf(format,argarray)
```

参数	描述
format	必需。转换格式。
argarray	必需。带有参数的一个数组，这些参数会被插到 format 字符串中的 % 符号处。

说明

参数 *format* 是转换的格式，以百分比符号 ("%") 开始到转换字符结束。下面的可能的 *format* 值：

- %% - 返回百分比符号
- %b - 二进制数
- %c - 依照 ASCII 值的字符
- %d - 带符号十进制数
- %e - 可续计数法（比如 1.5e+3）
- %u - 无符号十进制数
- %f - 浮点数(local settings aware)
- %F - 浮点数(not local settings aware)
- %o - 八进制数
- %s - 字符串
- %x - 十六进制数（小写字母）
- %X - 十六进制数（大写字母）

提示和注释

注释：如果 % 符号多于 *arg* 参数，则您必须使用占位符。占位符插到 % 符号后面，由数字和 "$" 组成。请参见例子 3。

提示： 相关函数： fprintf()、 printf()、 sprintf()、 vfprintf() 以及 vprintf()。

例子

例子 1

```
<?php
$str = "Hello";
$number = 123;
$txt = vsprintf("%s world. Day number %u",array($str,$number));
echo $txt;
?>
```

输出：

```
Hello world. Day number 123
```

例子 2

```
<?php
$num1 = 123;
$num2 = 456;
$txt = vsprintf("%f%f",array($num1,$num2));
echo $txt;
?>
```

输出：

```
123.000000456.000000
```

例子 3

使用占位符：

```
<?php
$number = 123;
$txt = vsprintf("With 2 decimals: %1\$.2f
<br />With no decimals: %1\$u",array($number));
echo $txt;
?>
```

输出：

```
With 2 decimals: 123.00
With no decimals: 123
```

PHP wordwrap() 函数

定义和用法

wordwrap() 函数按照指定长度对字符串进行折行处理。

如果成功，则返回折行后的字符串。如果失败，则返回 false。

语法

```
wordwrap(string,width,break,cut)
```

参数	描述
string	必需。规定要进行折行的字符串。
width	可选。规定最大行宽度。默认是 75。
break	可选。规定作为分隔符使用的字符（字串断开字符）。默认是 "\n"。
cut	可选。规定是否对大约指定宽度的单词进行折行。默认是 FALSE (no-wrap)。

例子

例子 1

```
<?php
$str = "An example on a long word is: Supercalifragulistic";
echo wordwrap($str,15);
?>
```

浏览器输出：

```
An example on a long word is: Supercalifragulistic
```

HTML 源代码：

```
<html>
<body>
An example on a
long word is:
Supercalifragulistic
</body>
</html>
```

例子 2

```
<?php
$str = "An example on a long word is: Supercalifragulistic";
echo wordwrap($str,15,"<br />\n");
?>
```

输出：

```
An example on a
long word is:
Supercalifragulistic
```

例子 3

```
<?php
$str = "An example on a long word is: Supercalifragulistic";
echo wordwrap($str,15,"<br />\n",TRUE);
?>
```

输出：

```
An example on a
long word is:
Supercalifragul
istic
```

PHP XML Parser 函数

PHP XML Parser 简介

XML 函数允许我们解析 XML 文档，但无法对其进行验证。

XML 是一种用于标准结构化文档交换的数据格式。您可以在我们的 XML 教程 中找到更多有关 XML 的信息。

该扩展使用 Expat XML 解析器。

Expat 是一种基于事件的解析器，它把 XML 文档视为一系列事件。当某个事件发生时，它调用一个指定的函数处理它。

Expat 是无验证的解析器，忽略任何链接到文档的 DTD。但是，如果文档的形式不好，则会以一个错误消息结束。

由于它基于事件，且无验证，Expat 具有快速并适合 web 应用程序的特性。

XML 解析器函数允许我们创建 XML 解析器，并为 XML 事件定义句柄。

安装

XML 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP XML Parser 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
utf8_decode()	把 UTF-8 字符串解码为 ISO-8859-1。	3
utf8_encode()	把 ISO-8859-1 字符串编码为 UTF-8。	3
xml_error_string()	获取 XML 解析器的错误描述。	3
xml_get_current_byte_index()	获取 XML 解析器的当前字节索引。	3
xml_get_current_column_number()	获取 XML 解析器的当前列号。	3
xml_get_current_line_number()	获取 XML 解析器的当前行号。	3
xml_get_error_code()	获取 XML 解析器错误代码。	3
xml_parse()	解析 XML 文档。	3
xml_parse_into_struct()	把 XML 数据解析到数组中。	3
xml_parser_create_ns()	创建带有命名空间支持的 XML 解析器。	4
xml_parser_create()	创建 XML 解析器。	3
xml_parser_free()	释放 XML 解析器。	3
xml_parser_get_option()	从 XML 解析器获取选项设置信息。	3
xml_parser_set_option()	为 XML 解析进行选项设置。	3
xml_set_character_data_handler()	建立字符数据处理器。	3
xml_set_default_handler()	建立默认的数据处理器。	3
xml_set_element_handler()	建立起始和终止元素处理器。	3
xml_set_end_namespace_decl_handler()	建立终止命名空间声明处理器。	4
xml_set_external_entity_ref_handler()	建立外部实体处理器。	3
xml_set_notation_decl_handler()	建立注释声明处理器。	3
xml_set_object()	在对象中使用 XML 解析器。	4
xml_set_processing_instruction_handler()	建立处理指令（PI）处理器。	3
xml_set_start_namespace_decl_handler()	建立起始命名空间声明处理器。	4
xml_set_unparsed_entity_decl_handler()	建立未解析实体定义声明处理器。	3

PHP XML Parser 常量

Constant
XML_ERROR_NONE (integer)
XML_ERROR_NO_MEMORY (integer)
XML_ERROR_SYNTAX (integer)
XML_ERROR_NO_ELEMENTS (integer)
XML_ERROR_INVALID_TOKEN (integer)
XML_ERROR_UNCLOSED_TOKEN (integer)
XML_ERROR_PARTIAL_CHAR (integer)
XML_ERROR_TAG_MISMATCH (integer)
XML_ERROR_DUPLICATE_ATTRIBUTE (integer)
XML_ERROR_JUNK_AFTER_DOC_ELEMENT (integer)
XML_ERROR_PARAM_ENTITY_REF (integer)
XML_ERROR_UNDEFINED_ENTITY (integer)
XML_ERROR_RECURSIVE_ENTITY_REF (integer)
XML_ERROR_ASYNC_ENTITY (integer)
XML_ERROR_BAD_CHAR_REF (integer)
XML_ERROR_BINARY_ENTITY_REF (integer)
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF (integer)
XML_ERROR_MISPLACED_XML_PI (integer)
XML_ERROR_UNKNOWN_ENCODING (integer)
XML_ERROR_INCORRECT_ENCODING (integer)
XML_ERROR_UNCLOSED_CDATA_SECTION (integer)
XML_ERROR_EXTERNAL_ENTITY_HANDLING (integer)
XML_OPTION_CASE_FOLDING (integer)
XML_OPTION_TARGET_ENCODING (integer)
XML_OPTION_SKIP_TAGSTART (integer)
XML_OPTION_SKIP_WHITE (integer)

PHP utf8_decode() 函数

定义和用法

utf8_decode() 函数把 UTF-8 字符串解码为 ISO-8859-1。

该函数把用 UTF-8 方式编码的 ISO-8859-1 字符串转换成单字节的 ISO-8859-1 字符串。

如果成功，该函数将返回解码字符串；否则返回 false。

语法

```
utf8_decode(string)
```

参数	描述
string	必需。规定要解码的字符串。

PHP utf8_encode() 函数

定义和用法

utf8_encode() 函数把 ISO-8859-1 字符串编码为 UTF-8。

Unicode 是全球标准，已经发展到能够通过唯一的编码来描述所有语言中的字符，外加大量的符号。

不过，并不是总能可靠地在计算机之间传递 Unicode 字符。UTF-8 可用于在计算机之间传输 Unicode 字符。

如果成功，该函数将返回编码字符串；否则返回 false。

语法

```
utf8_encode(string)
```

参数	描述
string	必需。规定要编码的字符串。

PHP xml_error_string() 函数

定义和用法

xml_error_string() 函数获取 XML 解析器的错误描述。

语法

```
xml_error_string(errorcode)
```

参数	描述
errorcode	必需。规定要使用的错误代码。该错误码是 xml_get_error_code() 函数的返回值。

说明

返回与 *errorcode* 描述的错误代码参数对应的文本描述字符串，若没有与之对应的描述，则返回 false。

例子

```
<?php
//无效的 xml 文件
$xmlfile = 'test.xml';

$xmlparser = xml_parser_create();

// open a file and read data
$fp = fopen($xmlfile, 'r');
while ($xmldata = fread($fp, 4096))
  {
  // parse the data chunk
  if (!xml_parse($xmlparser,$xmldata,feof($fp)))
    {
    die( print "ERROR: "
    . xml_error_string(xml_get_error_code($xmlparser))
    . "<br />"
    . "Line: "
    . xml_get_current_line_number($xmlparser)
    . "<br />"
    . "Column: "
    . xml_get_current_column_number($xmlparser)
    . "<br />");
    }
  }

xml_parser_free($xmlparser);
?>
```

输出：

```
ERROR: Mismatched tag
Line: 8
Column: 51
```

PHP xml_get_current_byte_index() 函数

定义和用法

xml_get_current_byte_index() 函数获取 XML 解析器的当前字节索引。

语法

```
xml_get_current_byte_index(parser)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。

说明

如果 *parser* 没有指向一个合法的解析器，该函数将返回 false，否则将返回解析器当前在其数据缓冲区中的字节索引（起始值为 0）。

例子

```
<?php
//无效的 xml 文件
$xmlfile = 'test.xml';

$xmlparser = xml_parser_create();

// 打开文件并读取数据
$fp = fopen($xmlfile, 'r');
while ($xmldata = fread($fp, 4096))
  {
  // parse the data chunk
  if (!xml_parse($xmlparser,$xmldata,feof($fp)))
    {
    die( print "ERROR: "
    . xml_error_string(xml_get_error_code($xmlparser))
    . "<br />"
    . "Line: "
    . xml_get_current_line_number($xmlparser)
    . "<br />"
    . "Column: "
    . xml_get_current_column_number($xmlparser)
    . "<br />"
    . "Byte Index: "
    . xml_get_current_byte_index($xmlparser)
    . "<br />");
    }
  }

xml_parser_free($xmlparser);
?>
```

输出：

```
ERROR: Mismatched tag
Line: 8
Column: 51
Byte Index: 96
```

PHP xml_get_current_line_number() 函数

定义和用法

xml_get_current_line_number() 函数获取 XML 解析器的当前行号。

语法

```
xml_get_current_line_number(parser)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。

说明

如果 *parser* 参数没有指向一个合法的解析器，该函数将返回 FALSE，否则将返回指定解析器在其缓存中的当前行号。

例子

```
<?php
//无效 xml 文件
$xmlfile = 'test.xml';

$xmlparser = xml_parser_create();

// 打开文件并读取数据
$fp = fopen($xmlfile, 'r');
while ($xmldata = fread($fp, 4096))
  {
  // parse the data chunk
  if (!xml_parse($xmlparser,$xmldata,feof($fp)))
    {
    die( print "ERROR: "
    . xml_error_string(xml_get_error_code($xmlparser))
    . "<br />"
    . "Line: "
    . xml_get_current_line_number($xmlparser)
    . "<br />"
    . "Column: "
    . xml_get_current_column_number($xmlparser)
    . "<br />");
    }
  }

xml_parser_free($xmlparser);
?>
```

输出：

```
ERROR: Mismatched tag
Line: 8
Column: 61
```

PHP xml_get_error_code() 函数

定义和用法

xml_get_error_code() 函数获取 XML 解析器错误代码。

如果成功，则返回错误代码。否则，返回 false。

语法

```
xml_get_error_code(parser)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。

例子

```
<?php
//无效的 xml 文件
$xmlfile = 'test.xml';

$xmlparser = xml_parser_create();

// 打开文件并读取数据
$fp = fopen($xmlfile, 'r');
while ($xmldata = fread($fp, 4096))
  {
  // parse the data chunk
  if (!xml_parse($xmlparser,$xmldata,feof($fp)))
    {
    die( print "ERROR: "
    . xml_get_error_code($xmlparser)
    . "<br />"
    . "Line: "
    . xml_get_current_line_number($xmlparser)
    . "<br />"
    . "Column: "
    . xml_get_current_column_number($xmlparser)
    . "<br />");
    }
  }

xml_parser_free($xmlparser);
?>
```

输出：

```
ERROR: 76
Line: 8
Column: 61
```

PHP xml_parse() 函数

定义和用法

xml_parse() 函数解析 XML 文档。

如果成功，则返回 true。否则，返回 false。

语法

```
xml_parse(parser,xml,end)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
xml	必需。规定要解析的 XML 数据。
end	可选。如果该参数是 true，则 xml 参数中的数据为当前解析中最后一段数据。

提示和注释

提示：要创建 XML 解析器，请使用 xml_parser_create() 函数。

例子

```
<?php
//无效的 xml 文件
$xmlfile = 'test.xml';

$xmlparser = xml_parser_create();

// 打开文件并读取数据
$fp = fopen($xmlfile, 'r');
while ($xmldata = fread($fp, 4096))
  {
  // parse the data chunk
  if (!xml_parse($xmlparser,$xmldata,feof($fp)))
    {
    die( print "ERROR: "
    . xml_get_error_code($xmlparser)
    . "<br />"
    . "Line: "
    . xml_get_current_line_number($xmlparser)
    . "<br />"
    . "Column: "
    . xml_get_current_column_number($xmlparser)
    . "<br />");
    }
  }

xml_parser_free($xmlparser);
?>
```

输出：

```
ERROR: 76
Line: 8
Column: 61
```

PHP xml_parse_into_struct() 函数

定义和用法

xml_parse_into_struct() 函数把 XML 数据解析到数组中。

该函数把 XML 数据解析到 2 个数组中：

- Value 数组 - 包含来自被解析的 XML 的数据
- Index 数组 - 包含指向Value 数组中值的位置的指针

如果成功，则该函数返回 1。否则返回 0。

语法

```
xml_parse_into_struct(parser,xml,value_arr,index_arr)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
xml	必需。规定要解析的 XML 数据。
value_arr	必需。规定 XML 数据的目标数组。
index_arr	可选。规定 index 数据的目标数组。

提示和注释

注释：xml_parse_into_struct() 若失败返回 0，成功则返回 1。这和 false 与 true 不同，使用例如 === 运算符时要注意。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php
//无效 xml 文件
$xmlfile = 'test.xml';
$xmlparser = xml_parser_create();

// 打开文件并读取数据
$fp = fopen($xmlfile, 'r');
$xmldata = fread($fp, 4096);

xml_parse_into_struct($xmlparser,$xmldata,$values);

xml_parser_free($xmlparser);
print_r($values);
?>
```

输出：

```
Array
(
[0] => Array
  (
  [tag] => NOTE
  [type] => open
  [level] => 1
  [value] =>
  )
  [1] => Array
    (
    [tag] => TO
    [type] => complete
    [level] => 2
    [value] => George
    )
[2] => Array
  (
  [tag] => NOTE
  [value] =>
  [type] => cdata
  [level] => 1
  )
  [3] => Array
    (
    [tag] => FROM
    [type] => complete
    [level] => 2
    [value] => John
    )
[4] => Array
  (
  [tag] => NOTE
  [value] =>
  [type] => cdata
  [level] => 1
  )
  [5] => Array
    (
    [tag] => HEADING
    [type] => complete
    [level] => 2
    [value] => Reminder
    )
[6] => Array
  (
  [tag] => NOTE
  [value] =>
```

```
    [type] => cdata
    [level] => 1
    )
    [7] => Array
      (
      [tag] => BODY
      [type] => complete
      [level] => 2
      [value] => Don't forget the meeting!
      )
  [8] => Array
    (
    [tag] => NOTE
    [value] =>
    [type] => cdata
    [level] => 1
    )
  [9] => Array
    (
    [tag] => NOTE
    [type] => close
    [level] => 1
    )
  )
```

PHP xml_parser_create_ns() 函数

定义和用法

xml_parser_create_ns() 函数创建带有命名空间支持的 XML 解析器。

该函数建立一个新的 XML 解析器并返回可被其它 XML 函数使用的资源句柄。

语法

```
xml_parser_create_ns(encoding,separator)
```

参数	描述
encoding	可选。规定输出编码。
encoding	可选。规定标签名和命名空间的输出分隔符。默认是 ":"。

说明

可选参数 *encoding* 在 PHP 4 中用来指定要被解析的 XML 输入的字符编码方式。

PHP 5 开始，自动侦测输入的 XML 的编码，因此 *encoding* 参数仅用来指定解析后输出数据的编码。

在 PHP 4 中，默认输出的编码与输入数据的编码是相同的。如果传递了空字符串，解析器会尝试搜索头 3 或 4 个字节以确定文档的编码。

在 PHP 5.0.0 和 5.0.1 中，默认输出的字符编码是 ISO-8859-1，而 PHP 5.0.2 及以上版本是 UTF-8。

解析器支持的编码有 ISO-8859-1, UTF-8 和 US-ASCII。

提示和注释

提示：要释放 XML 解析器，请使用 xml_parser_free() 函数。

提示：要创建没有命名空间支持的 XML 解析器，请使用 xml_parser_create() 函数。

例子

```
<?php
$xmlparser = xml_parser_create_ns();

xml_parser_free($xmlparser);
?>
```

PHP xml_parser_create() 函数

定义和用法

xml_parser_create() 函数创建 XML 解析器。

该函数建立一个新的 XML 解析器并返回可被其它 XML 函数使用的资源句柄。

语法

```
xml_parser_create(encoding)
```

参数	描述
encoding	可选。规定输出编码。

说明

可选参数 *encoding* 在 PHP 4 中用来指定要被解析的 XML 输入的字符编码方式。

PHP 5 开始，自动侦测输入的 XML 的编码，因此 *encoding* 参数仅用来指定解析后输出数据的编码。

在 PHP 4 中，默认输出的编码与输入数据的编码是相同的。如果传递了空字符串，解析器会尝试搜索头 3 或 4 个字节以确定文档的编码。

在 PHP 5.0.0 和 5.0.1 中，默认输出的字符编码是 ISO-8859-1，而 PHP 5.0.2 及以上版本是 UTF-8。

解析器支持的编码有 ISO-8859-1, UTF-8 和 US-ASCII。

提示和注释

提示：要释放 XML 解析器，请使用 xml_parser_free() 函数。

提示：要创建带有命名空间支持的 XML 解析器，请使用 xml_parser_create_ns() 函数。

例子

```
<?php
$xmlparser = xml_parser_create();

xml_parser_free($xmlparser);
?>
```

PHP xml_parser_free() 函数

定义和用法

xml_parser_free() 函数释放 XML 解析器。

如果成功，则返回 true。否则返回 false。

语法

```
xml_parser_free(parser)
```

参数	描述
parser	必需。规定要释放的 XML 解析器。

提示和注释

提示：要创建 XML 解析器，请使用 xml_parser_create() 函数。

例子

```
<?php
$xmlparser = xml_parser_create();

xml_parser_free($xmlparser);
?>
```

PHP xml_parser_get_option() 函数

定义和用法

xml_parser_get_option() 函数从 XML 解析器获取选项设置信息。

语法

```
xml_parser_get_option(parser,option)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
option	必需。规定要获取的设置选项名称。可能的值： XML_OPTION_CASE_FOLDING XML_OPTION_TARGET_ENCODING

说明

如果 *parser* 参数没有指向一个合法的解析器或者 *option* 参数无效，该函数将返回 FALSE（同时产生 E_WARNING 警告）。否则将返回指定设置选项的值。

例子

```
<?php
$xmlparser = xml_parser_create();

echo xml_parser_get_option($xmlparser, XML_OPTION_CASE_FOLDING);

xml_parser_free($xmlparser);
?>
```

PHP xml_parser_set_option() 函数

定义和用法

xml_parser_set_option() 函数为 XML 解析器进行选项设置。

如果成功，则返回 true。如果失败，则返回 false。

语法

```
xml_parser_set_option(parser,option,value)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
option	必需。规定要设置的设置选项名称。可能的值： `XML_OPTION_CASE_FOLDING` `XML_OPTION_SKIP_TAGSTART` `XML_OPTION_SKIP_WHITE` `XML_OPTION_TARGET_ENCODING`
value	必需。规定选项的新值。

例子

```
<?php
$xmlparser = xml_parser_create();

xml_parser_set_option($xmlparser, XML_OPTION_SKIP_WHITE, 1);

xml_parser_free($xmlparser);
?>
```

PHP xml_set_character_data_handler() 函数

定义和用法

xml_set_character_data_handler() 函数建立字符数据处理器。

该函数规定当解析器在 XML 文件中找到字符数据时所调用的函数。

如果处理器被成功的建立，该函数将返回 true；否则返回 false。

语法

```
xml_set_character_data_handler(parser,handler)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
handler	必需。规定作为事件处理器使用的函数。

由 *handler* 参数规定的函数必须有两个参数：

参数	描述
parser	必需。规定一个变量，包含调用处理器的 XML 解析器。
data	必需。规定包含字符数据的变量。

说明

handler 参数也可以是一个数组，其中包含对象引用和方法名。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php

$parser=xml_parser_create();

function char($parser,$data)
  {
  echo $data;
  }

xml_set_character_data_handler($parser,"char");
$fp=fopen("test.xml","r");

while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

xml_parser_free($parser);
?>
```

输出：

```
George John Reminder Don't forget the meeting!
```

PHP xml_set_default_handler() 函数

定义和用法

xml_set_default_handler() 函数为 XML 解析器建立默认的数据处理器。

该函数规定在只要解析器在 XML 文件中找到数据时，所调用的函数。

如果处理器被成功的建立，该函数将返回 true；否则返回 false。

语法

```
xml_set_default_handler(parser,handler)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
handler	必需。规定作为事件处理器使用的函数。

由 *handler* 参数规定的函数必须有三个参数：

参数	描述
parser	必需。规定一个变量，包含调用处理器的 XML 解析器。
data	必需。规定包含数据的变量。

说明

handler 参数也可以是一个数组，其中包含对象引用和方法名。

例子

XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP 代码：

```
<?php

$parser=xml_parser_create();

function default($parser,$data)
  {
  echo $data;
  }

xml_set_default_handler($parser,"default");
$fp=fopen("test.xml","r");

while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

xml_parser_free($parser);
?>
```

输出：

```
George John Reminder Don't forget the meeting!
```

如果在浏览器中查看源代码，会看到下列 HTML：

```
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

PHP xml_set_element_handler() 函数

定义和用法

xml_set_element_handler() 函数建立起始和终止元素处理器。

如果处理器被成功的建立，该函数将返回 true；否则返回 false。

语法

```
xml_set_element_handler(parser,start,end)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
start	必需。规定在元素开始调用的函数。
end	必需。规定在元素结束调用的函数。

由 *start* 参数规定的函数必须有三个参数：

参数	描述
parser	必需。规定一个变量，包含调用处理器的 XML 解析器。
name	必需。规定一个变量，包含元素的名称，这个元素触发该函数。
data	必需。规定一个数组，包含元素属性。

由 *end* 参数规定的函数必须有三个参数：

参数	描述
parser	必需。规定一个变量，包含调用处理器的 XML 解析器。
name	必需。规定一个变量，包含元素的名称，这个元素触发该函数。

说明

start 和 *end* 参数也可以是一个数组，其中包含对象引用和方法名。

例子

```
<?php

$parser=xml_parser_create();

function start($parser,$element_name,$element_attrs)
  {
  switch($element_name)
    {
    case "NOTE":
    echo "-- Note --<br />";
    break;
    case "TO":
    echo "To: ";
    break;
    case "FROM":
    echo "From: ";
    break;
    case "HEADING":
    echo "Heading: ";
    break;
    case "BODY":
    echo "Message: ";
    }
  }

function stop($parser,$element_name)
  {
  echo "<br />";
  }

function char($parser,$data)
  {
  echo $data;
  }

xml_set_element_handler($parser,"start","stop");
xml_set_character_data_handler($parser,"char");
$fp=fopen("test.xml","r");

while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

xml_parser_free($parser);

?>
```

输出：

```
-- Note --
To: George
From: John
Heading: Reminder
Message: Don't forget the meeting!
```

PHP xml_set_external_entity_ref_handler() 函数

定义和用法

xml_set_external_entity_ref_handler() 函数规定当解析器在 XML 文档中找到外部实体时被调用的函数。

如果处理器被成功的建立，该函数将返回 true；否则返回 false。

语法

```
xml_set_external_entity_ref_handler(parser,handler)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
handler	必需。规定当解析器找到外部实体时被调用的函数。

由 *handler* 参数规定的函数必须有六个参数：

参数	描述
parser	必需。规定一个变量，包含调用处理器的 XML 解析器。
name	必需。规定包含外部实体名称的变量。
base	必需。规定一个变量，包含解析外部实体的系统标识符（system_id ）的基础。当前该参数通常都被设置为空字符串。
system_id	必需。规定包含外部实体的系统标识符的变量。
public_id	必需。规定包含外部实体的公共标识符的变量。

说明

handler 参数也可以是一个数组，其中包含对象引用和方法名。

例子

```
<?php

$parser=xml_parser_create();

function char($parser,$data)
  {
  echo $data;
  }

function ext_ent_handler($parser,$ent,$base,$sysID,$pubID)
  {
  echo "$ent";
  echo "$sysID";
  echo "$pubID";
  }

xml_set_character_data_handler($parser,"char");
xml_set_external_entity_ref_handler($parser, "ext_ent_handler");
$fp=fopen("test.xml","r");

while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

xml_parser_free($parser);

?>
```

PHP xml_set_notation_decl_handler() 函数

定义和用法

xml_set_notation_decl_handler() 函数规定当解析器在 XML 文档中找到符号声明时被调用的函数。

如果处理器被成功的建立，该函数将返回 true；否则返回 false。

注释："符号声明"，英文为 notation declaration，也有部分文献译为"注释声明"。

语法

```
xml_set_notation_decl_handler(parser,handler)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
handler	必需。规定当解析器找到符号声明时被调用的函数。

由 *handler* 参数规定的函数必须有六个参数：

参数	描述
parser	必需。规定一个变量，包含调用处理器的 XML 解析器。
name	必需。规定包含实体名称的变量。
base	必需。规定一个变量，包含解析实体的系统标识符（system_id ）的基础。当前该参数通常都被设置为空字符串。
system_id	必需。规定包含实体的系统标识符的变量。
public_id	必需。规定包含实体的公共标识符的变量。
notation	必需。规定一个变量，包含标识实体数据类型的符号。

说明

handler 参数也可以是一个数组，其中包含对象引用和方法名。

例子

```
<?php

$parser=xml_parser_create();

function char($parser,$data)
  {
  echo $data;
  }

function not_decl_handler($parser,$not,$base,$sysID,$pubID)
  {
  echo "$not<br />";
  echo "$sysID<br />";
  echo "$pubID<BR />";
  }

xml_set_character_data_handler($parser,"char");
xml_set_notation_decl_handler($parser, "not_decl_handler");
$fp=fopen("test.xml","r");

while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

xml_parser_free($parser);

?>
```

PHP xml_set_object() 函数

定义和用法

xml_set_object() 函数允许在对象中使用 XML 解析器。

语法

```
xml_set_object(parser,object)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
object	必需。规定设置解析器的对象。

说明

该函数使得 *parser* 指定的解析器可以被用在 *object* 对象中。所有的回叫函数（callback function）都可以由 xml*set_element_handler()* 等函数来设置，它们被假定为 *_object* 对象的方法。

例子

```
<?php
class XMLParser
{
var $xmlparser;

function XMLParser()
  {
  $this->xmlparser = xml_parser_create();
  xml_set_object($this->xmlparser, $this);
  xml_set_character_data_handler($this->xmlparser,"char");
  xml_set_element_handler($this->xmlparser, "start_tag","end_tag");
  }

function parse($data)
  {
  xml_parse($this->xmlparser, $data);
  }

function parse_File($xmlfile)
  {
  $fp = fopen($xmlfile, 'r');
  while ($xmldata = fread($fp, 4096))
    {
    if
    (!xml_parse($this->xmlparser, $xmldata))
      {
      //If error
      die( print "ERROR: "
      . xml_error_string(xml_get_error_code($this->xmlparser))
      . "<br />Line: "
      . xml_get_current_line_number($this->xmlparser)
      . "<br />Column: "
      . xml_get_current_column_number($this->xmlparser)
      . "<br />");
      }
    }
  }

function start_tag($xmlparser, $tag, $attributes)
  {
  print $tag . "<br />";
  }

function end_tag(){}

function char($xmlparser,$data)
  {
  echo $data . "<br />";
  }

function close_Parser()
  {
  xml_parser_free($this->xmlparser);
  }
}

$myxmlparser = new XMLParser();
$myxmlparser->parse_File("test.xml");
$myxmlparser->close_parser();

?>
```

PHP xml_set_processing_instruction_handler() 函数

定义和用法

xml_set_processing_instruction_handler() 函数规定当解析器在 XML 文档中找到处理指令时所调用的函数。

处理指令包含在 <? 和 ?> 分隔符中。

如果处理器被成功的建立，该函数将返回 true；否则返回 false。

例子：在本例中，处理指令把一个样式表与 XML 文档关联起来：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="default.xsl" type="text/xml"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

语法

```
xml_set_processing_instruction_handler(parser,handler)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
handler	必需。规定一个函数。

由 *handler* 参数规定的函数必须有三个参数：

参数	描述
parser	必需。规定一个变量，包含调用处理器的 XML 解析器。
target	必需。规定包含处理指令目标的变量。
data	必需。规定包含处理指令数据的变量。

说明

handler 参数也可以是一个数组，其中包含对象引用和方法名。

例子

```
<?php

$parser=xml_parser_create();

function char($parser,$data)
  {
  echo $data;
  }

function pi_handler($parser, $target, $data)
  {
  echo "Target: $target<br />";
  echo "Data: $data<br />";
  }

xml_set_character_data_handler($parser,"char");
xml_set_processing_instruction_handler($parser, "pi_handler");
$fp=fopen("test.xml","r");

while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

xml_parser_free($parser);

?>
```

PHP xml_set_unparsed_entity_decl_handler() 函数

定义和用法

xml_set_unparsed_entity_decl_handler() 函数规定在遇到无法解析的实体名称（NDATA）声明时被调用的函数。

如果处理器被成功的建立，该函数将返回 true；否则返回 false。

语法

```
xml_set_unparsed_entity_decl_handler(parser,handler)
```

参数	描述
parser	必需。规定要使用的 XML 解析器。
handler	必需。规定一个函数。

由 *handler* 参数规定的函数必须有六个参数：

参数	描述
parser	必需。规定一个变量，包含调用处理器的 XML 解析器。
name	必需。规定包含实体名称的变量。
base	必需。规定一个变量，包含解析实体的系统标识符（system_id ）的基础。当前该参数通常都被设置为空字符串。
system_id	必需。规定包含实体的系统标识符的变量。
public_id	必需。规定包含实体的公共标识符的变量。
notation	必需。规定一个变量，包含标识实体数据类型的符号。

说明

handler 参数也可以是一个数组，其中包含对象引用和方法名。

例子

```
<?php

$parser=xml_parser_create();

function char($parser,$data)
  {
  echo $data;
  }

function unparsed_ent_handler($parser,$entname,
$base,$sysID,$pubID,$notname)
  {
  print "$entname";
  print "$sysID";
  print "$pubID";
  print "$notname";
  }

xml_set_character_data_handler($parser,"char");
xml_set_unparsed_entity_decl_handler($parser,
"unparsed_ent_handler");

$fp=fopen("test.xml","r");

while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

xml_parser_free($parser);

?>
```

PHP Zip File 函数

PHP Zip File 简介

压缩文件函数允许我们读取压缩文件。

安装

如需在服务器上运行 Zip File 函数，必须安装这些库：

- Guido Draheim 的 ZZIPlib 库：下载 ZZIPlib 库
- Zip PELC 扩展：下载 Zip PELC 扩展

在 Linux 系统上安装

PHP 5+：_*Zip* 函数和 *Zip* 库默认不会启用，必须从上面的链接下载。请使用 --with-zip=DIR_ 配置选项来包含 Zip 支持。

在 Windows 系统上安装

_PHP 5+：_Zip 函数默认不会启用，必须从上面的链接下载 php_zip.dll 和 ZZIPlib 库。必须在php.ini 之内启用 php_zip.dll。

如需启用任何 PHP 扩展，PHP extension_dir 设置（在 php.ini 文件中）应该设置为该 PHP 扩展所在的目录。举例 extension_dir 的值可能是 c:\php\ext。

PHP Zip File 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
zip_close()	关闭 ZIP 文件。	4
zip_entry_close()	关闭 ZIP 文件中的一个项目。	4
zip_entry_compressedsize()	返回 ZIP 文件中的一个项目的被压缩尺寸。	4
zip_entry_compressionmethod()	返回 ZIP 文件中的一个项目的压缩方法。	4
zip_entry_filesize()	返回 ZIP 文件中的一个项目的实际文件尺寸。	4
zip_entry_name()	返回 ZIP 文件中的一个项目的名称。	4
zip_entry_open()	打开 ZIP 文件中的一个项目以供读取。	4
zip_entry_read()	读取 ZIP 文件中的一个打开的项目。	4
zip_open()	打开 ZIP 文件。	4
zip_read()	读取 ZIP 文件中的下一个项目。	4

PHP Zip File 常量

无。

PHP zip_close() 函数

定义和用法

zip_close() 函数关闭由 zip_open() 函数打开的 zip 档案文件。

语法

```
zip_close(zip)
```

参数	描述
zip	必需。规定要关闭的 zip 资源（由 zip_open() 打开的 zip 文件）。

例子

```
<?php
$zip = zip_open("test.zip");

zip_read($zip);

// 一些代码...

zip_close($zip);
?>
```

PHP zip_entry_close() 函数

定义和用法

zip_entry_close() 函数关闭由 zip_entry_open() 函数打开的 zip 档案文件。

语法

```
zip_entry_close(zip_entry)
```

参数	描述
zip_entry	必需。规定要关闭的 zip 项目资源（由 zip_read() 打开的 zip 项目）。

例子

```
<?php
$zip = zip_open("test.zip");

if ($zip)
  {
  while ($zip_entry = zip_read($zip))
    {
    echo "<p>";
    echo "Name: " . zip_entry_name($zip_entry) . "<br />";

    if (zip_entry_open($zip, $zip_entry))
      {
      // 一些代码...
      zip_entry_close($zip_entry);
      }
    echo "</p>";
  }
zip_close($zip);
}
?>
```

PHP zip_entry_compressedsize() 函数

定义和用法

zip_entry_compressedsize() 函数返回 zip 档案项目的压缩文件尺寸。

语法

```
zip_entry_compressedsize(zip_entry)
```

参数	描述
zip_entry	必需。规定要读取的 zip 项目资源（由 zip_read() 打开的 zip 项目）。

例子

```
<?php
$zip = zip_open("test.zip");

if ($zip)
  {
  while ($zip_entry = zip_read($zip))
    {
    echo "<p>";
    echo "Name: " . zip_entry_name($zip_entry) . "<br />";
    echo "Compressed Size: "
    . zip_entry_compressedsize($zip_entry);
    echo "</p>";
    }
  zip_close($zip);
  }
?>
```

输出：

```
Name: ziptest.txt
Compressed Size: 68

Name: htmlziptest.html
Compressed Size: 159
```

PHP zip_entry_compressionmethod() 函数

定义和用法

zip_entry_compressionmethod() 函数返回 zip 档案项目的压缩方法。

语法

```
zip_entry_compressionmethod(zip_entry)
```

参数	描述
zip_entry	必需。规定要读取的 zip 项目资源（由 zip_read() 打开的 zip 项目）。

例子

```
<?php
$zip = zip_open("test.zip");

if ($zip)
  {
  while ($zip_entry = zip_read($zip))
    {
    echo "<p>";
    echo "Name: " . zip_entry_name($zip_entry) . "<br />";
    echo "Compression Method: "
    . zip_entry_compressionmethod($zip_entry);
    echo "</p>";
    }
  zip_close($zip);
  }
?>
```

输出：

```
Name: ziptest.txt
Compression Method: deflated

Name: htmlziptest.html
Compression Method: deflated
```

PHP zip_entry_filesize() 函数

定义和用法

zip_entry_filesize() 函数返回 zip 档案项目的原始大小（在压缩之前）。

语法

```
zip_entry_filesize(zip_entry)
```

参数	描述
zip_entry	必需。规定要读取的 zip 项目资源（由 zip_read() 打开的 zip 项目）。

例子

```
<?php
$zip = zip_open("test.zip");

if ($zip)
  {
  while ($zip_entry = zip_read($zip))
    {
    echo "<p>";
    echo "Name: " . zip_entry_name($zip_entry) . "<br />";
    echo "Original size: " . zip_entry_filesize($zip_entry);
    echo "</p>";
    }
  zip_close($zip);
  }
?>
```

输出：

```
Name: ziptest.txt
Original size: 68

Name: htmlziptest.html
Original size: 159
```

PHP zip_entry_name() 函数

定义和用法

zip_entry_name() 函数返回 zip 档案项目的名称。

语法

```
zip_entry_name(zip_entry)
```

参数	描述
zip_entry	必需。规定要读取的 zip 项目资源（由 zip_read() 打开的 zip 项目）。

例子

```
<?php
$zip = zip_open("test.zip");

if ($zip)
  {
  while ($zip_entry = zip_read($zip))
    {
    echo "Name: " . zip_entry_name($zip_entry) . "<br />";
    }
  zip_close($zip);
  }
?>
```

输出类似：

```
Name: ziptest.txt
Name: htmlziptest.html
```

PHP zip_entry_open() 函数

定义和用法

zip_entry_open() 函数打开一个 ZIP 档案项目以供读取。

语法

```
zip_entry_open(zip,zip_entry,mode)
```

参数	描述
zip	必需。规定要读取的 zip 资源（由 zip_open() 打开的 zip 文件）。
zip_entry	必需。规定要打开的 zip 项目资源（由 zip_read() 打开的 zip 项目）。
mode	可选。规定 zip 档案项目的访问类型。

说明

在 PHP 5 中，mode 会被忽略，且总为 "rb"。这是因为在 PHP 中的 zip 支持是只读的。

例子

```
<?php
$zip = zip_open("test.zip");

if ($zip)
  {
  while ($zip_entry = zip_read($zip))
    {
    echo "<p>";
    echo "Name: " . zip_entry_name($zip_entry) . "<br />";

    if (zip_entry_open($zip, $zip_entry))
      {
      // some code
      }
    echo "</p>";
  }

zip_close($zip);

}
?>
```

PHP zip_entry_read() 函数

定义和用法

zip_entry_read() 函数从打开的 zip 档案项目中获取内容。

如果成功，则返回项目的内容。如果失败，则返回 false。

语法

```
zip_entry_read(zip_entry,length)
```

参数	描述
zip_entry	必需。规定要读取的 zip 项目资源（由 zip_read() 打开的 zip 项目）。
length	可选。规定返回的字节数。默认是 1024。

例子

```
<?php
$zip = zip_open("test.zip");

if ($zip)
  {
  while ($zip_entry = zip_read($zip))
    {
    echo "<p>";
    echo "Name: " . zip_entry_name($zip_entry) . "<br />";

    if (zip_entry_open($zip, $zip_entry))
      {
      echo "File Contents:<br/>";
      $contents = zip_entry_read($zip_entry);
      echo "$contents<br />";
      zip_entry_close($zip_entry);
      }
    echo "</p>";
  }

zip_close($zip);

}
?>
```

PHP zip_open() 函数

定义和用法

zip_open() 函数打开 ZIP 文件以供读取。

如果成功，则返回 zip 文件档案资源。如果失败，则返回 false。

语法

```
zip_open(filename)
```

参数	描述
filename	必需。规定要打开的 zip 文件的文件名和路径。

提示和注释

提示：新打开的 zip 文件资源之后可被 zip_read() 和 zip_close() 函数使用。

例子

```
<?php
$zip = zip_open("test.zip");

zip_read($zip);

// 一些代码...

zip_close($zip);
?>
```

PHP zip_read() 函数

定义和用法

zip_read() 函数读取打开的 zip 档案中的下一个文件。

如果成功，则返回包含 zip 档案中一个文件的资源。如果没有更多的项目可供读取，则返回 false。

语法

```
zip_read(zip)
```

参数	描述
zip	必需。规定要读取的 zip 资源（由 zip_open() 打开的 zip 文件）。

提示和注释

提示：由 zip_read() 函数返回的资源可供 zip_entry... 类的函数使用。

例子

```
<?php
$zip = zip_open("test.zip");

zip_read($zip);

// 一些代码...

zip_close($zip);
?>
```

PHP 杂项函数

PHP 杂项函数简介

我们把不属于其他类别的函数归纳到这个页面。

安装

杂项函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

Runtime 配置

杂项函数函数的行为受到 php.ini 中设置的影响。

杂项函数配置选项：

名称	默认	描述	可更改
ignore_user_abort	"0"	FALSE 指示只要脚本在客户机终止连接后尝试进行输出，脚本将被终止。	PHP_INI_ALL
highlight.string	"#DD0000"	供突出显示符合 PHP 语法的字符串而使用的颜色。	PHP_INI_ALL
highlight.comment	"#FF8000"	供突出显示 PHP 注释而使用的颜色。	PHP_INI_ALL
highlight.keyword	"#007700"	供突出显示 PHP 关键词而使用的颜色（比如圆括号和分号）。	PHP_INI_ALL
highlight.bg	"#FFFFFF"	背景颜色。	PHP_INI_ALL
highlight.default	"#0000BB"	PHP 语法的默认颜色。	PHP_INI_ALL
highlight.html	"#000000"	HTML 代码的颜色。	PHP_INI_ALL
browscap	NULL	浏览器性能文件的名称和位置（例如：browscap.ini）。	PHP_INI_SYSTEM

PHP 杂项函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
connection_aborted()	检查是否断开客户机。	3
connection_status()	返回当前的连接状态。	3
connection_timeout()	在 PHP 4.0.5 中不赞成使用。	3
constant()	返回一个常量的值。	4
define()	定义一个常量。	3
defined()	检查某常量是否存在。	3
die()	输出一条消息，并退出当前脚本。	3
eval()	把字符串按照 PHP 代码来计算。	3
exit()	输出一条消息，并退出当前脚本。	3
get_browser()	返回用户浏览器的性能。	3
highlight_file()	对文件进行语法高亮显示。	4
highlight_string()	对字符串进行语法高亮显示。	4
ignore_user_abort()	设置与客户机断开是否会终止脚本的执行。	3
pack()	把数据装入一个二进制字符串。	3
php_check_syntax()	在 PHP 5.0.5 中不赞成使用。	5
php_strip_whitespace()	返回已删除 PHP 注释以及空白字符的源代码文件。	5
show_source()	highlight_file() 的别名。	4
sleep()	延迟代码执行若干秒。	3
time_nanosleep()	延迟代码执行若干秒和纳秒。	5
time_sleep_until()	延迟代码执行指定的时间。	5
uniqid()	生成唯一的 ID。	3
unpack()	从二进制字符串对数据进行解包。	3
usleep()	延迟代码执行若干微秒。	3

PHP Date / Time 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
CONNECTION_ABORTED		
CONNECTION_NORMAL		
CONNECTION_TIMEOUT		
COMPILER_HALT_OFFSET	5	

PHP connection_aborted() 函数

定义和用法

connection_aborted() 函数检查是否断开客户机。

如果已终止连接，则该函返回 1，否则返回 0。

语法

```
connection_aborted()
```

例子

创建一个函数，在客户机终止脚本时写入一条日志消息：

```
<?php
function check_abort()
  {
  if (connection_aborted())
  error_log ("Script $GLOBALS[SCRIPT_NAME]" .
  "$GLOBALS[SERVER_NAME] was aborted by the user.");
  }

//要执行的一些代码

// 在脚本结束时调用 check_abort 函数
register_shutdown_function("check_abort");
?>
```

PHP connection_status() 函数

定义和用法

connection_status() 函数返回当前的连接状态。

可返回的可能值：

- 0 - CONNECTION_NORMAL - 连接运行正常
- 1 - CONNECTION_ABORTED - 连接由用户或网络错误终止
- 2 - CONNECTION_TIMEOUT - 连接超时
- 3 - CONNECTION_ABORTED & CONNECTION_TIMEOUT

语法

```
connection_status()
```

例子

```
<?php

switch (connection_status())
{
case CONNECTION_NORMAL:
   $txt = 'Connection is in a normal state';
   break;
case CONNECTION_ABORTED:
   $txt = 'Connection aborted';
   break;
case CONNECTION_TIMEOUT:
   $txt = 'Connection timed out';
   break;
case (CONNECTION_ABORTED & CONNECTION_TIMEOUT):
   $txt = 'Connection aborted and timed out';
   break;
default:
   $txt = 'Unknown';
   break;
}

echo $txt;
?>
```

PHP constant() 函数

定义和用法

constant() 函数返回常量的值。

语法

```
constant(constant)
```

参数	描述
constant	必需。规定要检查的常量的名称。

提示和注释

注释：该函数仅适用于 class 常量。

例子

```
<?php
//定义一个常量
define("GREETING","Hello world!");

echo constant("GREETING");
?>
```

输出：

```
Hello world!
```

PHP define() 函数

定义和用法

define() 函数定义一个常量。

常量类似变量，不同之处在于：

- 在设定以后，常量的值无法更改
- 常量名不需要开头的美元符号 ($)
- 作用域不影响对常量的访问
- 常量值只能是字符串或数字

语法

```
define(_name_,_value_,_case_insensitive_)
```

参数	描述
name	必需。规定常量的名称。
value	必需。规定常量的值。
case_insensitive	可选。规定常量的名称是否对大小写敏感。若设置为 true，则对大小写不敏感。默认是 false（大小写敏感）。

例子

例子 1

定义一个大小写敏感的常量：

```
<?php
define("GREETING","Hello world!");
echo constant("GREETING");
?>
```

输出：

```
Hello world!
```

例子 2

定义一个大小写不敏感的常量：

```
<?php
define("GREETING","Hello world!",TRUE);
echo constant("greeting");
?>
```

输出：

```
Hello world!
```

PHP defined() 函数

定义和用法

defined() 函数检查某常量是否存在。

若常量存在，则返回 true，否则返回 false。

语法

```
defined(name)
```

参数	描述
name	必需。规定要检查的常量的名称。

例子

```
<?php
define("GREETING","Hello world!");
echo defined("GREETING");
?>
```

输出：

```
1
```

PHP die() 函数

定义和用法

die() 函数输出一条消息，并退出当前脚本。

该函数是 exit() 函数的别名。

语法

```
die(status)
```

参数	描述
status	必需。规定在退出脚本之前写入的消息或状态号。状态号不会被写入输出。

说明

如果 *status* 是字符串，则该函数会在退出前输出字符串。

如果 *status* 是整数，这个值会被用作退出状态。退出状态的值在 0 至 254 之间。退出状态 255 由 PHP 保留，不会被使用。状态 0 用于成功地终止程序。

提示和注释

注释：如果 PHP 的版本号大于等于 4.2.0，那么在 *status* 是整数的情况下，不会输出该参数。

例子

```
<?php
$site = "http://www.w3school.com.cn/";
fopen($site,"r")
or die("Unable to connect to $site");
?>
```

PHP eval() 函数

定义和用法

eval() 函数把字符串按照 PHP 代码来计算。

该字符串必须是合法的 PHP 代码，且必须以分号结尾。

如果没有在代码字符串中调用 return 语句，则返回 NULL。如果代码中存在解析错误，则 eval() 函数返回 false。

语法

```
eval(phpcode)
```

参数	描述
phpcode	必需。规定要计算的 PHP 代码。

提示和注释

注释：返回语句会立即终止对字符串的计算。

注释：该函数对于在数据库文本字段中供日后计算而进行的代码存储很有用。

例子

```
<?php
$string = "beautiful";
$time = "winter";

$str = 'This is a $string $time morning!';
echo $str. "<br />";

eval("\$str = \"$str\";");
echo $str;
?>
```

输出：

```
This is a $string $time morning!
This is a beautiful winter morning!
```

PHP exit() 函数

定义和用法

exit() 函数输出一条消息，并退出当前脚本。

该函数是 die() 函数的别名。

语法

```
exit(status)
```

参数	描述
status	必需。规定在退出脚本之前写入的消息或状态号。状态号不会被写入输出。

说明

如果 *status* 是字符串，则该函数会在退出前输出字符串。

如果 *status* 是整数，这个值会被用作退出状态。退出状态的值在 0 至 254 之间。退出状态 255 由 PHP 保留，不会被使用。状态 0 用于成功地终止程序。

提示和注释

注释：如果 PHP 的版本号大于等于 4.2.0，那么在 *status* 是整数的情况下，不会输出该参数。

例子

```
<?php
$site = "http://www.w3school.com.cn/";
fopen($site,"r")
or exit("Unable to connect to $site");
?>
```

PHP get_browser() 函数

定义和用法

get_browser() 函数返回用户浏览器的性能。

该函数通过查阅用户的 browscap.ini 文件，来测定用户浏览器的性能。

若成功，则该函数返回包含用户浏览器信息的一个对象或一个数组，若失败，则返回 false。

语法

```
get_browser(user_agent,return_array)
```

参数	描述
user_agent	可选。规定 HTTP 用户代理的名称。默认是 $HTTP_USER_AGENT 的值。您可以通过设置 NULL 绕过该参数。
return_array	可选。如果该参数设置为 true，本函数会返回一个数组而不是对象。

提示和注释

注释：返回语句会立即终止对字符串的计算。

注释：该函数对于在数据库文本字段中供日后计算而进行的代码存储很有用。

例子

```
<?php
echo $_SERVER['HTTP_USER_AGENT'] . "<br /><br />";
$browser = get_browser(null,true);
print_r($browser);
?>
```

输出：

```
Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)

Array
(
[browser_name_regex] => ^mozilla/.\.0
(compatible; msie 6\.0.*;.*windows nt 5\.1.*\.net clr.*).*$
[browser_name_pattern] => Mozilla/?.0
(compatible; MSIE 6.0*;*Windows NT 5.1*.NET CLR*)*
[parent] => IE 6.0
[platform] => WinXP
[netclr] => 1
[browser] => IE
[version] => 6.0
[majorver] => 6
[minorver] => 0
[css] => 2
[frames] => 1
[iframes] => 1
[tables] => 1
[cookies] => 1
[backgroundsounds] => 1
[vbscript] => 1
[javascript] => 1
[javaapplets] => 1
[activexcontrols] => 1
[cdf] => 1
[aol] =>
[beta] =>
[win16] =>
[crawler] =>
[stripper] =>
[wap] =>
[ak] =>
[sk] =>
)
```

PHP highlight_file() 函数

定义和用法

highlight_file() 函数对文件进行语法高亮显示。

语法

```
highlight_file(filename,return)
```

参数	描述
filename	必需。要进行高亮处理的 PHP 文件的路径。
return	可选。如果设置 true，则本函数返回高亮处理的代码。

说明

本函数通过使用 PHP 语法高亮程序中定义的颜色，输出或返回包含在 *filename* 中的代码的语法高亮版本。

许多服务器被配置为对带有 *phps* 后缀的文件进行自动高亮处理。例如，在查看 example.phps 时，将显示该文件被语法高亮显示的源代码。要启用该功能，请把下面这一行添加到 *httpd.conf*：

```
AddType application/x-httpd-php-source .phps
```

返回值

如果 *return* 参数被设置为 true，那么该函数会返回被高亮处理的代码，而不是输出它们。否则，若成功，则返回 true，失败则返回 false。

提示和注释

警告：需要注意的是，在使用 highlight_file() 函数时，请不要因为疏忽而泄露诸如密码或其他类型的敏感信息，否则会出现潜在的安全风险。

例子

"test.php":

```
<html>
<body>
<?php
highlight_file("test.php");
?>
</body>
</html>
```

输出：

```
<html>
<body>
<?php
highlight_file("test.php");
?>
</body>
</html>
```

在浏览器中查看的结果类似这样：

```
<html>
<body>
<code>
<span style="color: #000000">&lt;html&gt;
<br />
&lt;body&gt;
<br />
<span style="color: #0000BB">&lt;?php
<br />highlight_file</span>
<span style="color: #007700">(</span>
<span style="color: #DD0000">"test.php"</span>
<span style="color: #007700">);<br /></span>
<span style="color: #0000BB">?&gt;<br /></span>
&lt;/body&gt;
<br />
&lt;/html&gt;</span>
</code>
</body>
</html>
```

PHP highlight_string() 函数

定义和用法

highlight_string() 函数对字符串进行语法高亮显示。

语法

```
highlight_string(string,return)
```

参数	描述
string	必需。要进行高亮处理的字符串。
return	可选。如果设置 true，则本函数返回高亮处理的代码。

说明

本函数通过使用 PHP 语法高亮程序中定义的颜色，输出或返回给定的 PHP 代码的语法高亮版本。

返回值

如果 *return* 参数被设置为 true，那么该函数会以字符串返回被高亮处理的代码，而不是输出它们。否则，若成功，则返回 true，失败则返回 false。

例子

```
<html>
<body>
<?php
highlight_string("Hello world! <?php phpinfo(); ?>");
?>
</body>
</html>
```

输出：

```
Hello world! <?php phpinfo();?>
```

在浏览器中查看的结果类似这样：

```
<html>
<body>
 <code>
  <span style="color: #000000">Hello world! 
  <span style="color: #0000BB">&lt;?php phpinfo</span>
  <span style="color: #007700">();</span>
  <span style="color: #0000BB">?&gt;</span>
  </span>
 </code>
</body>
</html>
```

PHP ignore_user_abort() 函数

定义和用法

ignore_user_abort() 函数设置与客户机断开是否会终止脚本的执行。

本函数返回 user-abort 设置的之前的值（一个布尔值）。

语法

```
ignore_user_abort(setting)
```

参数	描述
setting	可选。如果设置为 true，则忽略与用户的断开，如果设置为 false，会导致脚本停止运行。如果未设置该参数，会返回当前的设置。

提示和注释

注释：PHP 不会检测到用户是否已断开连接，直到尝试向客户机发送信息为止。简单地使用 echo 语句无法确保信息发送，参阅 flush() 函数。

例子

```
<?php
ignore_user_abort();
?>
```

输出：

```
0
```

PHP pack() 函数

定义和用法

pack() 函数把数据装入一个二进制字符串。

语法

```
pack(format,args+)
```

参数	描述
format	必需。规定在包装数据时所使用的格式。
args+	可选。规定被包装的一个或多个参数。

format 参数的可能值：

- a - NUL-padded string
- A - SPACE-padded string
- h - Hex string, low nibble first
- H - Hex string, high nibble first
- c - signed char
- C - unsigned char
- s - signed short (always 16 bit, machine byte order)
- S - unsigned short (always 16 bit, machine byte order)
- n - unsigned short (always 16 bit, big endian byte order)
- v - unsigned short (always 16 bit, little endian byte order)
- i - signed integer (machine dependent size and byte order)
- I - unsigned integer (machine dependent size and byte order)
- l - signed long (always 32 bit, machine byte order)
- L - unsigned long (always 32 bit, machine byte order)
- N - unsigned long (always 32 bit, big endian byte order)
- V - unsigned long (always 32 bit, little endian byte order)
- f - float (machine dependent size and representation)
- d - double (machine dependent size and representation)
- x - NUL byte
- X - Back up one byte
- @ - NUL-fill to absolute position

例子

例子 1

```
<?php
echo pack("C3",80,72,80);
?>
```

输出：

```
PHP
```

例子 2

```
<?php
echo pack("C*",80,72,80);
?>
```

输出：

```
PHP
```

PHP strip_whitespace() 函数

定义和用法

strip_whitespace() 函数返回已删除 PHP 注释以及空白字符的源代码文件。

该函数对于检测脚本中的实际代码量很有用。

语法

```
strip_whitespace(filename)
```

参数	描述
filename	必需。规定文件名。

说明

若成功，则返回被剥离的源代码，若失败，则返回空字符串。

注释：在 PHP 5.0.1，该函数的行为与上面的描述一致。在这之前，它仅返回空字符串。

例子

"test.php":

```
<?php
// PHP comment

/*
*  Another PHP comment
*/

echo php_strip_whitespace      ("test.php");
?>
```

输出：

```
<?php
echo php_strip_whitespace ("test.php"); ?>
```

PHP show_source() 函数

定义和用法

show_source() 函数对文件进行语法高亮显示。

本函数是 highlight_file() 的别名。

语法

```
show_source(filename,return)
```

参数	描述
filename	必需。要进行高亮处理的 PHP 文件的路径。
return	可选。如果设置 true，则本函数返回高亮处理的代码。

说明

本函数通过使用 PHP 语法高亮程序中定义的颜色，输出或返回包含在 *filename* 中的代码的语法高亮版本。

许多服务器被配置为对带有 *phps* 后缀的文件进行自动高亮处理。例如，在查看 example.phps 时，将显示该文件被语法高亮显示的源代码。要启用该功能，请把下面这一行添加到 *httpd.conf*：

```
AddType application/x-httpd-php-source .phps
```

返回值

如果 *return* 参数被设置为 true，那么该函数会返回被高亮处理的代码，而不是输出它们。否则，若成功，则返回 true，失败则返回 false。

提示和注释

警告：需要注意的是，在使用 show_source() 函数时，请不要因为疏忽而泄露诸如密码或其他类型的敏感信息，否则会出现潜在的安全风险。

例子

"test.php":

```
<html>
<body>
<?php
show_source("test.php");
?>
</body>
</html>
```

输出：

```
<html>
<body>
<?php
show_source("test.php");
?>
</body>
</html>
```

在浏览器中查看的结果类似这样：

```
<html>
<body>
<code>
<span style="color: #000000">&lt;html&gt;
<br />
&lt;body&gt;
<br />
<span style="color: #0000BB">&lt;?php
<br />show_source</span>
<span style="color: #007700">(</span>
<span style="color: #DD0000">"test.php"</span>
<span style="color: #007700">);<br /></span>
<span style="color: #0000BB">?&gt;<br /></span>
&lt;/body&gt;
<br />
&lt;/html&gt;</span>
</code>
</body>
</html>
```

PHP sleep() 函数

定义和用法

sleep() 函数延迟代码执行若干秒。

语法

```
sleep(seconds)
```

参数	描述
seconds	必需。以秒计的暂停时间。

返回值

若成功，返回 0，否则返回 false。

错误／异常

如果指定的描述 seconds 是负数，该函数将生成一个 E_WARNING。

例子

```
<?php
echo date('h:i:s') . "<br />";

//暂停 10 秒
sleep(10);

//重新开始
echo date('h:i:s');

?>
```

输出：

```
12:00:08
12:00:18
```

PHP time_nanosleep() 函数

定义和用法

time_nanosleep() 函数延迟代码执行若干秒和纳秒。

语法

```
time_nanosleep(seconds,nanoseconds)
```

参数	描述
seconds	必需。必须是正整数。
nanoseconds	必需。必须是小于 10 亿的正整数。

说明

延迟程序执行指定的 *seconds* 和 *nanoseconds* 数。

返回值

如果成功则返回 TRUE，失败则返回 FALSE

如果延迟被一个信号中断，将返回带有以下组件的关联数组：

- seconds - 延迟中剩余的秒数
- nanoseconds - 延迟中剩余的纳秒数

提示和注释

注释：本函数未在 Windows 平台下实现。

例子

```
<?php
if (time_nanosleep(3,500000000) === true)
  {
  echo "暂停 3 秒半";
  }
?>
```

PHP time_sleep_until() 函数

定义和用法

time_sleep_until() 函数延迟代码执行直到指定的时间。

语法

```
time_sleep_until(timestamp)
```

参数	描述
timestamp	必需。脚本唤醒时的时间戳。

说明

使脚本暂停执行，直到指定的 *timestamp*。

返回值

如果成功则返回 TRUE，失败则返回 FALSE。

错误／异常

如果指定的时间戳位于过去，则该函数将生成一个 E_WARNING。

提示和注释

注释：所有信号都将在脚本唤醒后递送。

注释：本函数未在 Windows 平台下实现。

例子

```
<?php
// 从现在起 10 秒后唤醒
time_sleep_until(time()+10);
?>
```

PHP uniqid() 函数

定义和用法

uniqid() 函数基于以微秒计的当前时间，生成一个唯一的 ID。

语法

```
uniqid(prefix,more_entropy)
```

参数	描述
prefix	可选。为 ID 规定前缀。如果两个脚本恰好在相同的微秒生成 ID，该参数很有用。
more_entropy	可选。规定位于返回值末尾的更多的熵。

说明

如果 *prefix* 参数为空，则返回的字符串有 13 个字符串长。如果 *more_entropy* 参数设置为 true，则是 23 个字符串长。

如果 *more_entropy* 参数设置为 true，则在返回值的末尾添加额外的熵（使用组合线形同余数生成程序），这样可以结果的唯一性更好。

返回值

以字符串的形式返回唯一标识符。

提示和注释

注释：由于基于系统时间，通过该函数生成的 ID 不是最佳的。如需生成绝对唯一的 ID，请使用 md5() 函数（请在字符串函数参考中查找）。

例子

```
<?php
echo uniqid();
?>
```

输出类似：

```
4415297e3af8c
```

PHP unpack() 函数

定义和用法

unpack() 函数从二进制字符串对数据进行解包。

语法

```
unpack(format,data)
```

参数	描述
format	必需。规定在解包数据时所使用的格式。
data	可选。规定被解包的二进制数据。

format 参数的可能值：

- a - NUL-padded string
- A - SPACE-padded string
- h - Hex string, low nibble first
- H - Hex string, high nibble first
- c - signed char
- C - unsigned char
- s - signed short (always 16 bit, machine byte order)
- S - unsigned short (always 16 bit, machine byte order)
- n - unsigned short (always 16 bit, big endian byte order)
- v - unsigned short (always 16 bit, little endian byte order)
- i - signed integer (machine dependent size and byte order)
- I - unsigned integer (machine dependent size and byte order)
- l - signed long (always 32 bit, machine byte order)
- L - unsigned long (always 32 bit, machine byte order)
- N - unsigned long (always 32 bit, big endian byte order)
- V - unsigned long (always 32 bit, little endian byte order)
- f - float (machine dependent size and representation)
- d - double (machine dependent size and representation)
- x - NUL byte
- X - Back up one byte
- @ - NUL-fill to absolute position

例子

例子 1

```
<?php
$data = "PHP";
print_r(unpack("C*",$data));
?>
```

输出：

```
Array
(
[1] => 80
[2] => 72
[3] => 80
)
```

例子 2

```
<?php
$data = "PHP";
print_r(unpack("C*myint",$data));
?>
```

输出：

```
Array
(
[myint1] => 80
[myint2] => 72
[myint3] => 80
)
```

例子 3

```
<?php
$bin = pack("c2n2",0x1234,0x5678,65,66);
print_r(unpack("c2chars/n2int",$bin));
?>
```

输出：

```
Array
(
[chars1] => 52
[chars2] => 120
[int1] => 65
[int2] => 66
)
```

PHP usleep() 函数

定义和用法

usleep() 函数延迟代码执行若干微秒。

语法

```
usleep(microseconds)
```

参数	描述
microseconds	必需。以微秒计的暂停时间。

返回值

无返回值。

提示和注释

注释：在 PHP 5 之前，该函数无法工作于 Windows 系统上。

注释：一微秒等于百万分之一秒。

例子

```
<?php
echo date('h:i:s') . "<br />";

//延迟 10 描述
usleep(10000000);

//再次开始
echo date('h:i:s');
?>
```

输出：

```
09:23:14
09:23:24
```

PHP 5 时区

PHP 支持的时区

下面是 PHP 支持的时区的完整列表，这些对一些 PHP 日期函数很有用。

- 非洲
- 美洲
- 南极洲
- 北冰洋
- 亚洲
- 大西洋
- 大洋洲
- 欧洲
- 印度洋
- 太平洋

非洲

Africa/Abidjan	Africa/Accra	Africa/Addis_Ababa	Africa/Algiers
Africa/Asmera	Africa/Bamako	Africa/Bangui	Africa/Banjul
Africa/Blantyre	Africa/Brazzaville	Africa/Bujumbura	Africa/Cairo
Africa/Ceuta	Africa/Conakry	Africa/Dakar	Africa/Dar_es_Sala
Africa/Douala	Africa/El_Aaiun	Africa/Freetown	Africa/Gaborone
Africa/Johannesburg	Africa/Juba	Africa/Kampala	Africa/Khartoum
Africa/Kinshasa	Africa/Lagos	Africa/Libreville	Africa/Lome
Africa/Lubumbashi	Africa/Lusaka	Africa/Malabo	Africa/Maputo
Africa/Mbabane	Africa/Mogadishu	Africa/Monrovia	Africa/Nairobi
Africa/Niamey	Africa/Nouakchott	Africa/Ouagadougou	Africa/Porto-Novo
Africa/Timbuktu	Africa/Tripoli	Africa/Tunis	Africa/Windhoek

美洲

America/Adak	America/Anchorage	Ame

America/Antigua	America/Araguaina	America/Arge
America/Argentina/Catamarca	America/Argentina/ComodRivadavia	America/Arge
America/Argentina/Jujuy	America/Argentina/La_Rioja	America/Arge
America/Argentina/Rio_Gallegos	America/Argentina/Salta	America/Arge
America/Argentina/San_Luis	America/Argentina/Tucuman	America/Arge
America/Aruba	America/Asuncion	America/Atiko
America/Atka	America/Bahia	America/Bahi
America/Barbados	America/Belem	America/Beliz
America/Blanc-Sablon	America/Boa_Vista	America/Bogo
America/Boise	America/Buenos_Aires	America/Cam
America/Campo_Grande	America/Cancun	America/Cara
America/Catamarca	America/Cayenne	America/Cayr
America/Chicago	America/Chihuahua	America/Cora
America/Cordoba	America/Costa_Rica	America/Cres
America/Cuiaba	America/Curacao	America/Danr
America/Dawson	America/Dawson_Creek	America/Denv
America/Detroit	America/Dominica	America/Edm
America/Eirunepe	America/El_Salvador	America/Ense
America/Fort_Wayne	America/Fortaleza	America/Glac
America/Godthab	America/Goose_Bay	America/Gran
America/Grenada	America/Guadeloupe	America/Guat
America/Guayaquil	America/Guyana	America/Halifa
America/Havana	America/Hermosillo	America/India
America/Indiana/Knox	America/Indiana/Marengo	America/India
America/Indiana/Tell_City	America/Indiana/Vevay	America/India
America/Indiana/Winamac	America/Indianapolis	America/Inuvi
America/Iqaluit	America/Jamaica	America/Jujuy
America/Juneau	America/Kentucky/Louisville	America/Kent
America/Knox_IN	America/Kralendijk	America/La_F
America/Lima	America/Los_Angeles	America/Louis
America/Lower_Princes	America/Maceio	America/Mana
America/Manaus	America/Marigot	America/Marti

America/Matamoros	America/Mazatlan	America/Meno
America/Menominee	America/Merida	America/Metla
America/Mexico_City	America/Miquelon	America/Mon
America/Monterrey	America/Montevideo	America/Mont
America/Montserrat	America/Nassau	America/New_
America/Nipigon	America/Nome	America/Noro
America/North_Dakota/Beulah	America/North_Dakota/Center	America/North
America/Ojinaga	America/Panama	America/Pang
America/Paramaribo	America/Phoenix	America/Port-
America/Port_of_Spain	America/Porto_Acre	America/Porto
America/Puerto_Rico	America/Rainy_River	America/Rank
America/Recife	America/Regina	America/Reso
America/Rio_Branco	America/Rosario	America/Santa
America/Santarem	America/Santiago	America/Santo
America/Sao_Paulo	America/Scoresbysund	America/Shipr
America/Sitka	America/St_Barthelemy	America/St_Jo
America/St_Kitts	America/St_Lucia	America/St_T
America/St_Vincent	America/Swift_Current	America/Tegu
America/Thule	America/Thunder_Bay	America/Tijua
America/Toronto	America/Tortola	America/Vanc
America/Virgin	America/Whitehorse	America/Winn
America/Yakutat	America/Yellowknife	

南极洲

Antarctica/Casey	Antarctica/Davis	Antarctica/DumontDUrville	Antarctica/I
Antarctica/McMurdo	Antarctica/Palmer	Antarctica/Rothera	Antarctica/S
Antarctica/Vostok			

北冰洋

Arctic/Longyearbyen

亚洲

Asia/Aden	Asia/Almaty	Asia/Amman	Asia/Anadyr	
Asia/Aqtobe	Asia/Ashgabat	Asia/Ashkhabad	Asia/Baghdad	A
Asia/Baku	Asia/Bangkok	Asia/Beirut	Asia/Bishkek	A
Asia/Calcutta	Asia/Choibalsan	Asia/Chongqing	Asia/Chungking	A
Asia/Dacca	Asia/Damascus	Asia/Dhaka	Asia/Dili	A
Asia/Dushanbe	Asia/Gaza	Asia/Harbin	Asia/Hebron	A
Asia/Hong_Kong	Asia/Hovd	Asia/Irkutsk	Asia/Istanbul	A
Asia/Jayapura	Asia/Jerusalem	Asia/Kabul	Asia/Kamchatka	A
Asia/Kashgar	Asia/Kathmandu	Asia/Katmandu	Asia/Khandyga	A
Asia/Krasnoyarsk	Asia/Kuala_Lumpur	Asia/Kuching	Asia/Kuwait	A
Asia/Macau	Asia/Magadan	Asia/Makassar	Asia/Manila	A
Asia/Nicosia	Asia/Novokuznetsk	Asia/Novosibirsk	Asia/Omsk	A
Asia/Phnom_Penh	Asia/Pontianak	Asia/Pyongyang	Asia/Qatar	A
Asia/Rangoon	Asia/Riyadh	Asia/Saigon	Asia/Sakhalin	A
Asia/Seoul	Asia/Shanghai	Asia/Singapore	Asia/Taipei	A
Asia/Tbilisi	Asia/Tehran	Asia/Tel_Aviv	Asia/Thimbu	A
Asia/Tokyo	Asia/Ujung_Pandang	Asia/Ulaanbaatar	Asia/Ulan_Bator	A
Asia/Ust-Nera	Asia/Vientiane	Asia/Vladivostok	Asia/Yakutsk	A
Asia/Yerevan				

大西洋

Atlantic/Azores	Atlantic/Bermuda	Atlantic/Canary	Atlantic/Cape_Verde
Atlantic/Faroe	Atlantic/Jan_Mayen	Atlantic/Madeira	Atlantic/Reykjavik
Atlantic/St_Helena	Atlantic/Stanley		

大洋洲

Australia/ACT	Australia/Adelaide	Australia/Brisbane	Australia/Broke
Australia/Currie	Australia/Darwin	Australia/Eucla	Australia/Hobart
Australia/Lindeman	Australia/Lord_Howe	Australia/Melbourne	Australia/North
Australia/Perth	Australia/Queensland	Australia/South	Australia/Sydney
Australia/Victoria	Australia/West	Australia/Yancowinna	

欧洲

Europe/Amsterdam	Europe/Andorra	Europe/Athens	Europe/Belfast
Europe/Berlin	Europe/Bratislava	Europe/Brussels	Europe/Bucharest
Europe/Busingen	Europe/Chisinau	Europe/Copenhagen	Europe/Dublin
Europe/Guernsey	Europe/Helsinki	Europe/Isle_of_Man	Europe/Istanbul
Europe/Kaliningrad	Europe/Kiev	Europe/Lisbon	Europe/Ljubljana
Europe/Luxembourg	Europe/Madrid	Europe/Malta	Europe/Mariehamn
Europe/Monaco	Europe/Moscow	Europe/Nicosia	Europe/Oslo
Europe/Podgorica	Europe/Prague	Europe/Riga	Europe/Rome
Europe/San_Marino	Europe/Sarajevo	Europe/Simferopol	Europe/Skopje
Europe/Stockholm	Europe/Tallinn	Europe/Tirane	Europe/Tiraspol
Europe/Vaduz	Europe/Vatican	Europe/Vienna	Europe/Vilnius
Europe/Warsaw	Europe/Zagreb	Europe/Zaporozhye	Europe/Zurich

印度洋

Indian/Antananarivo	Indian/Chagos	Indian/Christmas	Indian/Cocos	Ind
Indian/Kerguelen	Indian/Mahe	Indian/Maldives	Indian/Mauritius	Indi
Indian/Reunion				

太平洋

Pacific/Apia	Pacific/Auckland	Pacific/Chatham	Pacific/Chuuk
Pacific/Efate	Pacific/Enderbury	Pacific/Fakaofo	Pacific/Fiji
Pacific/Galapagos	Pacific/Gambier	Pacific/Guadalcanal	Pacific/Guam
Pacific/Johnston	Pacific/Kiritimati	Pacific/Kosrae	Pacific/Kwajalein
Pacific/Marquesas	Pacific/Midway	Pacific/Nauru	Pacific/Niue
Pacific/Noumea	Pacific/Pago_Pago	Pacific/Palau	Pacific/Pitcairn
Pacific/Ponape	Pacific/Port_Moresby	Pacific/Rarotonga	Pacific/Saipan
Pacific/Tahiti	Pacific/Tarawa	Pacific/Tongatapu	Pacific/Truk
Pacific/Wallis	Pacific/Yap		

免责声明

W3School提供的内容仅用于培训。我们不保证内容的正确性。通过使用本站内容随之而来的风险与本站无关。W3School简体中文版的所有内容仅供测试，对任何法律问题及风险不承担任何责任。